# bspyclone.bvp.bvp

index
d:\work\productioncode\clones\biosppy\bspyclone\bvp\bvp.py

```
.. module:: bvp
   :platform: Unix, Windows
   :synopsis: This module provides various functions to handle BVP signals.

.. moduleauthor:: Filipe Canento
```

## Modules

| | | | |
|---|---|---|---|
| bspyclone.filt | os | bspyclone.plux | bspyclone.tools |
| glob | bspyclone.peakd | scipy | unittest |
| numpy | pylab | bspyclone.sync | |

## Functions

**features**(Signal=None, SamplingRate=1000.0, Filter={})

```
    Retrieves relevant BVP signal features.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): sampling frequency (Hz).

        Filter (dict):  filter parameters.

    Kwrvals:
        Signal (array): output filtered signal (see notes 1).

        Amplitude (array): signal pulses amplitudes (in the units of the input signal).(TODO)

        Onset (array): indexes (or instants in seconds, see notes 2.b) of the pulses onsets.

        Peak (array): indexes (or instants in seconds, see notes 2.b) of the pulses peaks. (TODO)

        DicroticNotch (array): indexes (or instants in seconds, see notes 2.b) of the pulses dicrotic notchs. (TODO)

        IBI (array): Inter-Beat Intervals in msec (see notes 2.a).

        HR (array): Instantaneous Heart Rates in b.p.m. (see notes 2.a).

        mean (float): mean

        std (float): standard deviation

        var (float): variance

        skew (ndarry): skewness

        kurtosis (array): kurtosis

        ad (float): absolute deviation

    Configurable fields:
    {"name": "bvp.features", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "Amplitude", "Onset", "Pe

    See Also:
            filt

            pulse

            tls.statsf

    Notes:
        1 - If a filter is given as a parameter, then the returned keyworded values dict has a 'Signal' key.

        2 - If the sampling rate is defined, then:
            a) the returned keyworded values dict has keys 'IBI' and 'HR'.
            b) keys 'onset', 'peak', and 'DicroticNotch' are converted to instants of occurrence in seconds.

    Example:
        bvp = ...
        SamplingRate = ...
        res = pulse(Signal=bvp, SamplingRate=SamplingRate)

    References:
        .. [1]
```

**filt**(Signal=None, SamplingRate=1000.0, UpperCutoff=8.0, LowerCutoff=1.0, Order=4.0)

```
    Filters an input BVP signal.

    If only input signal is provide, it returns the filtered
    signal assuming a 1000Hz sampling frequency and the default
    filter parameters: low-pass filter with cutoff frequency of 8Hz
    followed by a high-pass filter with cutoff frequency of 1Hz.

    Kwargs:
        Signal (array): input signal.
```

SamplingRate (float): sampling frequency (Hz).

UpperCutoff (float): Low-pass filter cutoff frequency (Hz).

LowerCutoff (float): High-pass filter cutoff frequency (Hz).

Order (int): Filter order.

Kwrvals:
Signal (array): output filtered signal.

Configurable fields:{"name": "bvp.filt", "config": {"UpperCutoff": "8.", "SamplingRate": "1000.", "LowerCutoff": "1.", "Order": "4."}, "input

See Also:
flt.zpdfr

Notes:

Example:

References:
.. [1]

**onset**(Signal=None, SamplingRate=1000.0)
Determines the onsets of the BVP signal pulses.
Skips very corrupted signal parts.

Kwargs:
Signal (array): input signal.

SamplingRate (float): sampling frequency (Hz).

Kwrvals:
Onset (array):

Configurable fields:{"name": "bvp.onset", "config": {"SamplingRate": "1000."}, "inputs": ["Signal"], "outputs": ["Onset"]}

See Also:

Notes:

Example:

References:
.. [1]

**pulse**(Signal=None, SamplingRate=1000.0, Filter={})
Determines BVP signal pulse information.

Kwargs:
Signal (array): input signal.

SamplingRate (float): sampling frequency (Hz).

Filter (dict): filter parameters.

Kwrvals:
Signal (array): output filtered signal (see notes 1).

Amplitude (array): signal pulses amplitudes (in the units of the input signal). (TODO)

Onset (array): indexes (or instants in seconds, see notes 2.b) of the pulses onsets.

Peak (array): indexes (or instants in seconds, see notes 2.b) of the pulses peaks. (TODO)

DicroticNotch (array): indexes (or instants in seconds, see notes 2.b) of the pulses dicrotic notchs. (TODO)

IBI (array): Inter-Beat Intervals in msec (see notes 2.a).

HR (array): Instantaneous Heart Rates in b.p.m. (see notes 2.a).

Configurable fields:{"name": "bvp.pulse", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "Amplitu

See Also:
filt

onset

Notes:
1 - If a filter is given as a parameter, then the returned keyworded values dict has a 'Signal' key.

2 - If the sampling rate is defined, then:
a) the returned keyworded values dict has keys 'IBI' and 'HR'.
b) TODO: keys 'onset', 'peak', and 'DicroticNotch' are converted to instants of occurrence in seconds.

Example:
bvp = ...
SamplingRate = ...
res = pulse(Signal=bvp, SamplingRate=SamplingRate)

References:
.. [1]

# bspyclone.database.biomesh

index
d:\work\productioncode\clones\biosppy\bspyclone\database\biomesh.py

```
.. module:: biomesh
   :platform: Unix, Windows
   :synopsis: This module provides an API to use the BioMESH specification. See the specification at http://camoes.lx.it.pt/MediaWiki/index.php/Database_S

.. moduleauthor:: Carlos Carreiras
```

## Modules

| | | |
|---|---|---|
| bspyclone.database.h5db | pymongo | string |
| numpy | re | sympy |
| os | bspyclone.database.syncdb | warnings |

## Classes

biomesh
dataContainer
dataSelector
experiments
records
subjects

class **biomesh**

```
Class to operate on a BioMESH DB database.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__init__**(self, dbName=None, host='localhost', port=27017, dstPath='~', srvPath='/BioMESH', sync=False, altSync=False)

```
Establish a connection to a BioMESH DB server. If the database (DB) does not exist, one is created with the necessary basic structures.

Kwargs:
    dbName (str): Name of the DB to connect to.

    host (str): Network address of the MongoDB server. Default: 'localhost'.

    port (int): Port the MongoDB server is listening on. Default: 27017.

    dstPath (str): Path to store the HDF5 files. Default: '~'.

    srvPath (str): Path to store the HDF5 files on the remote server. Default: '/BioMESH'.

    sync (bool): Flag to perform synchronization with remote server. Default: True.

    altSync (bool): New experimental synchronization framework. Default: False.

Kwrvals:


See Also:


Notes:
    If the 'sync' flag is True, the connection to the database may take longer to establish due to the synchronization step. This is esp

Example:
    db = biomesh(dbName='biomesh_tst', host='193.136.222.234', port=27017, dstPath='~/tmp/biomesh', srvPath='/biomesh_tst', sync=False)

References:
    .. [1]
```

**close**(self)

```
Close the connection to the database.

Kwargs:


Kwrvals:
```

See Also:

Notes:

Example:
db.close()

References:
.. [1]

**drop**(self, flag=True)
Function to remove the database from the server. The HDF5 files are preserved.

Kwargs:
flag (bool): Flag to override user confirmation about removal. Set to False to avoid user prompt. Default: True.

Kwrvals:

See Also:

Notes:

Example:
db.drop()

References:
.. [1]

**expsInSub**(self, subjectId)
List all the experiments in the database belonging to a given subject.

Kwargs:
subjectId (int): The ID of the subject.

Kwrvals:
nameList (list): List with the names of the experiments in the database.

See Also:

Notes:

Example:
expList = db.expsInSub(0)['nameList']

References:
.. [1]

**h5Add**(self, filePath=None)
Procedure to add already created HDF5 files (records) to MongoDB.

Kwargs:
filePath (str): Location of the file to add.

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

**metaH5Add**(self, filePath=None)
Procedure to add an already created meta HDF5 file (experiments and subjects) to MongoDB.

Kwargs:
filePath (str): Location of the file to add.

Kwrvals:

See Also:

Notes:

Example:

References:

```
        .. [1]
```

**subsInExp**(self, experimentName)

```
    List all the subjects in the database belonging to a given experiment.

    Kwargs:
        experimentName (str): The name of the experiment.

    Kwrvals:
        idList (list): List with the IDs of the subjects.

    See Also:


    Notes:


    Example:
        subList = db.subsInExp('unknown')['idList']

    References:
        .. [1]
```

**syncNow**(self)

```
    Function to perform the synchronization with the remote server.

    Kwargs:
        None

    Kwrvals:
        None

    See Also:


    Notes:


    Example:
        db.sync()

    References:
        .. [1]
```

class **dataContainer**

```
Data container for signals and events.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__init__**(self, case=None, mdata={}, signal=[], timeStamps=[], values=[])

```
    Initialize the container according to case (signal or events).

    Kwargs:
        case (str): Case ('signals' or 'events') to store.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**__str__**(self)

```
    str operator.

    Kwargs:
```

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

class **dataSelector**

Wrapper for operator overloading to records.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

Methods defined here:

**__getitem__**(self, key)

x.__getitem__(y) <==> x[y]

Kwargs:
key (str, int, slice, list): Item to retrieve.

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

**__init__**(self, recordsInst, recordId, dataType='/')

Initiate the class.

Kwargs:
recordsInst (biomesh.records): Instance of the records collection.

recordId (int): ID of the record.

dataType (str): Type of the data. Default: '/'.

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

**__iter__**(self)

Iterator operator.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

**__len__**(self)
Number of datasets in the record.

x.__len__() <==> len(x)

Kwargs:

Kwrvals:

See Also:

Notes:
If the case (signal or events) is still unspecified, returns the total number of datasets. Otherwise, returns the number of datasets

Example:

References:
.. [1]

**__str__**(self)
str operator.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

**list**(self)
List the datasets.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

class **experiments**
Class to operate on the Experiments collection.

Kwargs:

Kwrvals:

See Also:

Notes:

```
Example:


References:
    .. [1]
```

Methods defined here:

**__init__**(self, db=None, meta=None, parent=None, **kwargs)

```
Initialize the experiments class.

Kwargs:
    db (pymongo.database.Database): An instance representing the DB.

    meta (str): The path to the meta HDF5 file.

Kwrvals:


See Also:


Notes:
    For experiments, the key 'name' is the preferred indexer.

Example:


References:
    .. [1]
```

**add**(self, experiment=None, flag=True)

```
To add an experiment to the DB's 'experiments' collection.

Kwargs:
    experiment (dict): Experiment (JSON) to add.

    flag (bool): Flag to store in HDF5. Default: True.

Kwrvals:
    experimentId (int): ID of the experiment in the DB.

See Also:


Notes:
    If the experiment already exists (i.e. there is an experiment with the same name) the ID of the experiment in the DB is returned.

Example:
    expId = db.experiments.add({'name': 'experiment'})['experimentId']

References:
    .. [1]
```

**get**(self, refine={}, restrict={})

```
Make a general query the 'experiments' collection.

Kwargs:
    refine (dict): Dictionary to refine the search. Default: {}.

    restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

Kwrvals:
    docList (list): The list of documents with the results of the query. Empty list if no match is found.

See Also:


Notes:


Example:
    # get all experiments
    res = db.experiments.get()['docList']
    # get experiments with 'field' set to 'new'
    res = db.experiments.get(refine={'field': 'new'})['docList']
    # get experiments with 'field' set to 'new' and 'flag' set to False
    res = db.experiments.get(refine={'field': 'new', 'flag': False})['docList']

References:
    .. [1]
```

**getById**(self, experimentId=None, restrict={})

```
Query the 'experiments' collection by ID.

Kwargs:
    experimentId (int): ID of the experiment to query.

    restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

Kwrvals:
    doc (dict): The document with the results of the query. None if no match is found.

See Also:
```

```
    Notes:


    Example:
        # get everything
        doc = db.experiments.getById(0)['doc']
        # only return the ID
        doc = db.experiments.getById(0, restrict={'_id': 1})['doc']
        # only return the name (ID is also returned!)
        doc = db.experiments.getById(0, restrict={'name': 1})['doc']
        # only return the name and don't return the ID
        doc = db.experiments.getById(0, restrict={'name': 1, '_id': 0})['doc']

    References:
        .. [1]
```

**getByName**(self, experimentName=None, restrict={})

```
    Query the 'experiments' collection by name.

    Kwargs:
        experimentName (str): Name of the experiment to query.

        restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

    Kwrvals:
        doc (dict): The document with the results of the query. None if no match in found.

    See Also:


    Notes:


    Example:
        # get everything
        doc = db.experiments.getByName('experiment')['doc']
        # only return the ID
        doc = db.experiments.getByName('experiment', restrict={'_id': 1})['doc']
        # only return the name (ID is also returned!)
        doc = db.experiments.getByName('experiment', restrict={'name': 1})['doc']
        # only return the name and don't return the ID
        doc = db.experiments.getByName('experiment', restrict={'name': 1, '_id': 0})['doc']

    References:
        .. [1]
```

**list**(self)

```
    List all the experiments in the database.

    Kwargs:


    Kwrvals:
        nameList (list): List with the names of the experiments in the database.

    See Also:


    Notes:


    Example:
        expList = db.experiments.list()['nameList']

    References:
        .. [1]
```

**update**(self, experimentName=None, info={})

```
    Update an experiment with the given information. Fields can be added, and its type changed, but not deleted.

    Kwargs:
        experimentName (str): Name of the experiment to update.

        info (dict): Dictionary with the information to update. Default: {}.

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.experiments.update('experiment', {'new': 'field'})

    References:
        .. [1]
```

## class **records**

```
Class to operate on the Records collection.
```

```
Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__getitem__**(self, key)

```
To get Records from the DB.

x.__getitem__(y) <==> x[y]

Kwargs:
    key (int, slice, list): Items to retrieve.

Kwrvals:


See Also:


Notes:


Example:
    container = db.records[0]
    containerList = db.records[:]
    containerList = db.records[[0, 3, 4]]

References:
    .. [1]
```

**__init__**(self, db=None, path=None, srvPath=None, meta=None, parent=None, **kwargs)

```
Initialize the records class.

Kwargs:
    db (pymongo.database.Database): An instance representing the DB.

    path (str): The path to store the HDF5 files.

    srvPath (str): The path to store the HDF5 files on the remote server.

    meta (str): The path to the meta HDF5 file.

Kwrvals:


See Also:


Notes:
    For records, the key '_id' is the preferred indexer.

Example:


References:
    .. [1]
```

**__iter__**(self)

```
Iterator operator over the Records in the DB.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:
    for item in db.records:
        signal = item['signals']['test'][0].signal
        metadataS = item['signals']['test'][0].metadata
        timeStamps = item['events']['test'][0].timeStamps
        values = item['events']['test'][0].values
        metadataE = item['events']['test'][0].metadata

References:
```

```
        .. [1]
```

**__len__**(self)

```
    Returns the number of records on the DB.

    x.__len__() <==> len(x)

    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:
        len(db.records)

    References:
        .. [1]
```

**add**(self, record=None, flag=True)

```
    To add a record to the DB's 'records' collection.

    Kwargs:
        record (dict): Record (JSON) to add.

        flag (bool): Flag to store in HDF5. Default: True.

    Kwrvals:
        recordId (int): ID of the record in the DB.

    See Also:


    Notes:
        A new record is always added, regardless of the existence of records with the same name. If the record to add has no 'experiment' ar

    Example:
        recId = db.records.add({'name': 'record'})['recordId']

    References:
        .. [1]
```

**addAudit**(self)

```
    To add information to 'audit' field.

    TO DO!

    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**addEvent**(self, recordId=None, eventType='/', timeStamps=None, values=None, mdata={}, flag=True, compress=False)

```
    To add events (asynchronous data) to a record.

    Kwargs:
        recordId (int): ID of the record.

        eventType (str): Type of the events to add. Default: '/'.

        timeStamps (array): Array of time stamps. Default: [].

        values (array): Array with data for each time stamp. Default: [].

        mdata (dict): JSON with metadata about the events. Default: {}.

        flag (bool): Flag to store in HDF5. Default: True.

        compress (bool): Flag to compress the data (GZIP). Default: False.

    Kwrvals:
        recordId (int): ID of the record.

        eventRef (str): Storge name of the events.

        eventType (str): Type of the inserted events.
```

```
    See Also:


    Notes:


    Example:
        res = db.records.addEvent(0, '/test', [0, 1, 2], [[0, 1], [2, 3], [4, 5]], {'comments': 'test event'})

    References:
        .. [1]
```

**addSignal**(self, recordId=None, signalType='/', signal=None, mdata={}, flag=True, compress=False, updateDuration=False)

```
    To add signal (synchronous) data to a record.

    Kwargs:
        recordId (int): ID of the record.

        signalType (str): Type of the signal to add. Default: '/'.

        signal (array): Signal to add. Default: [].

        mdata (dict): Dictionary (JSON) with metadata about the signal. Default: {}.

        flag (bool): Flag to store in HDF5. Default: True.

        compress (bool): Flag to compress the data (GZIP). Default: False.

        updateDuration (bool): Flag to update the duration of the record. Default: False.

    Kwrvals:
        recordId (int): ID of the record.

        signalRef (str): Storge name of the signal.

        signalType (str): Type of the inserted signal.

    See Also:


    Notes:


    Example:
        res = db.records.addSignal(0, '/test', [0, 1, 2, 3], {'comments': 'test signal'})

    References:
        .. [1]
```

**addTags**(self, recordId=None, tags=None, flag=True)

```
    Add tags to a record.

    Kwargs:
        recordId (int): ID of the record.

        tags (list): Tags to add.

        flag (bool): Flag to store in HDF5. Default: True.

    Kwrvals:
        None

    See Also:


    Notes:


    Example:
        db.records.addTags(0, ['a', 'b', 'c'])

    References:
        .. [1]
```

**delEvent**(self, recordId=None, eventType='/', eventRef=None)

```
    Remove events from a record.

    Kwargs:
        recordId (int): ID of the record.

        eventType (str): Type of the desired event. Default: '/'.

        eventRef (str, int): Storage name (global) or index (local) of the desired events.

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.records.delEvent(0, '/test', 0)

    References:
```

```
        .. [1]
```

**delEventType**(self, recordId=None, eventType='/')

```
    Remove an event type from a record (including all sub-types).

    Kwargs:
        recordId (int): ID of the record.

        eventType (str): Type of the desired event. Default: '/'.

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.records.delEventType(0, '/test')

    References:
        .. [1]
```

**delSignal**(self, recordId=None, signalType='/', signalRef=None)

```
    Remove a signal (synchronous data) from a record.

    Kwargs:
        recordId (int): ID of the record.

        signalType (str): Type of the desired data. Default: '/'.

        signalRef (str, int): Storge name (global) or index (local) of the desired signal.

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.records.delSignal(0, '/test', 0)

    References:
        .. [1]
```

**delSignalType**(self, recordId=None, signalType='/')

```
    Remove a signal type from a record (including all sub-types).

    Kwargs:
        recordId (int): ID of the record.

        signalType (str): Type of the desired data. Drfault: '/'

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.records.delSignalType(0, '/test')

    References:
        .. [1]
```

**delTags**(self, recordId=None, tags=None)

```
    Delete tags from a record.

    Kwargs:
        recordId (int): ID of the record.

        tags (list): Tags to add.

    Kwrvals:
        None

    See Also:


    Notes:


    Example:
        db.records.delTags(0, ['a', 'b', 'c'])

    References:
        .. [1]
```

**delete**(self, recordId=None, keepFile=True)

```
Remove a record from the database.

Kwargs:
    recordId (int): ID of the record.

    keepFile (bool): Flag to keep local file. Default: True.

Kwrvals:


See Also:


Notes:


Example:
    db.records.delete(0)

References:
    .. [1]
```

**get**(self, refine={}, restrict={})

```
 Make a general query the 'records' collection.

Kwargs:
    refine (dict): Dictionary to refine the search. Default: {}.

    restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

Kwrvals:
    docList (list): The list of documents with the results of the query. Empty list if no match is found.

See Also:


Notes:


Example:
    # get all records
    res = db.records.get()['docList']
    # get records with 'field' set to 'new'
    res = db.records.get(refine={'field': 'new'})['docList']
    # get records with 'field' set to 'new' and 'flag' set to False
    res = db.records.get(refine={'field': 'new', 'flag': False})['docList']

References:
    .. [1]
```

**getAll**(self, restrict={}, count=-1, randomFlag=False)

```
Generate a list of records present in the DB. The list may include all records, records pertaining to an experiment (or list of experime

Kwargs:
    restrict (dict): To restrict the results. Can have the following keys:
        experiment (int, str, list): Experiment ID, experiment name, list of experiment IDs, list of experiment names, or list of exper
        subject (int, str, list): Subject ID, subject name, list of subject IDs, list of subject names, or list of experiment names and
        Default: {}

    count (int): The resulting list has, at most, 'count' items. Set to -1 to output all records found. Default: -1.

    randomFlag (bool): Set this flag to True to randomize the output list. Default: False.

Kwrvals:
    idList (list): List with the records' IDs that match the search.

See Also:


Notes:


Example:
    # get all records
    idList = db.records.getAll()['idList']
    # get at most 5 records
    idList = db.records.getAll(count=5)['idList']
    # get all records from experiment 'exp'
    idList = db.records.getAll(restrict={'experiment': 'exp'})['idList']
    # get all records from experiment 'exp' and 'dxp'
    idList = db.records.getAll(restrict={'experiment': ['exp', 'dxp']})['idList']
    # get all records from subject 0
    idList = db.records.getAll(restrict={'subject': 0})['idList']
    # get all records from subject 0 and 1
    idList = db.records.getAll(restrict={'subject': [0, 1]})['idList']
    # get all records from subject 0 and 1 and 'John Smith'
    idList = db.records.getAll(restrict={'subject': [0, 1, 'John Smith']})['idList']
    # get all records from experiment 'exp', but only from subjects 0 and 1
    idList = db.records.getAll(restrict={'experiment': 'exp', 'subject': [0, 1]})['idList']

References:
    .. [1]
```

**getById**(self, recordId=None, restrict={})

```
Query the 'records' collection by ID.
```

```
    Kwargs:
        recordId (int): ID of the record.

        restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

    Kwrvals:
        doc (dict): The document with the results of the query. None if no match in found.

    See Also:


    Notes:


    Example:
        # get everything
        doc = db.records.getById(0)['doc']
        # only return the ID
        doc = db.records.getById(0, restrict={'_id': 1})['doc']
        # only return the name (ID is also returned!)
        doc = db.records.getById(0, restrict={'name': 1})['doc']
        # only return the name and don't return the ID
        doc = db.records.getById(0, restrict={'name': 1, '_id': 0})['doc']

    References:
        .. [1]
```

**getByName**(self, recordName=None, restrict={})

```
    Query the 'records' collection by name.

    Kwargs:
        recordName (str): Name of the record.

        restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

    Kwrvals:
        doc (dict): The document with the results of the query. None if no match in found.

    See Also:


    Notes:


    Example:
        # get everything
        doc = db.records.getByName('record')['doc']
        # only return the ID
        doc = db.records.getByName('record', restrict={'_id': 1})['doc']
        # only return the name (ID is also returned!)
        doc = db.records.getByName('record', restrict={'name': 1})['doc']
        # only return the name and don't return the ID
        doc = db.records.getByName('record', restrict={'name': 1, '_id': 0})['doc']

    References:
        .. [1]
```

**getEvent**(self, recordId=None, eventType='/', eventRef=None)

```
    Retrieve events (asynchronous data) from a record.

    Kwargs:
        recordId (int): ID of the record.

        eventType (str): Type of the desired event. Default: '/'.

        eventRef (str, int): Storage name (global) or index (local) of the desired events.

    Kwrvals:
        timeStamps (array): Array of time stamps.

        values (array): Array with data for each time stamp.

        mdata (dict): Dictionary with metadata about the events.

    See Also:


    Notes:


    Example:
        res = db.records.getEvent(0, '/test', 0)
        timeStamps = res['timeStamps']
        values = res['values']
        metadata = res['mdata']

    References:
        .. [1]
```

**getSignal**(self, recordId=None, signalType='/', signalRef=None)

```
    Retrive a signal(synchronous data) from a record and corresponding metadata.

    Kwargs:
        recordId (int): ID of the record.

        signalType (str): Type of the desired signal. Default: '/'.
```

```
        signalRef (int, str): Storge name (global) or index (local) of the desired signal.

    Kwrvals:
        signal (array): Array with the signal.

        mdata (dict): Dictionary with the signal's accompanying metadata.

    See Also:


    Notes:


    Example:
        res = db.records.getSignal(0, '/test', 0)
        signal = res['signal']
        metadata = res['mdata']

    References:
        .. [1]
```

**listAndTags**(self, tags=None)

```
    Lists the records that simultaneously have all the given tags (AND operator).

    Kwargs:
        tags (list): Tags to match.

    Kwrvals:
        idList (list): List with the records' IDs that match the search.

    See Also:


    Notes:


    Example:
        res = db.records.listAndTags(['a', 'b', 'c'])['idList']

    References:
        .. [1]
```

**listNotTags**(self, tags=None)

**listOrTags**(self, tags=None)

```
    Lists the records that have, at least, one of the given tags (OR operator).

    Kwargs:
        tags (list): Tags to match.

    Kwrvals:
        idList (list): List with the records' IDs that match the search.

    See Also:


    Notes:


    Example:
        res = db.records.listOrTags(['a', 'b', 'c'])['idList']

    References:
        .. [1]
```

**listSymbolicTags**(self, query=None)

**listTypes**(self, recordId=None)

```
    To list the types of signals and events.

    Kwargs:
        recordId (int): ID of the record.

    Kwrvals:
        signalTypes (list): List of data types.

        eventTypes (list): List of event types.

    See Also:


    Notes:


    Example:
        res = db.records.listTypes(0)

    References:
        .. [1]
```

**update**(self, recordId=None, info={}, flag=True)

```
    Update a record with the given information. Fields can be added, and its type changed, but not deleted.

    Kwargs:
        recordId (int): ID of the record to update.
```

```
        info (dict): Dictionary with the information to update. Default: {}.

        flag (bool): Flag to store in HDF5. Default: True.

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.records.update(0, {'new': 'field'})

    References:
        .. [1]
```

class **subjects**

```
Class to operate on the Subjects collection.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__init__**(self, db=None, meta=None, parent=None, **kwargs)

```
    Initialize the subjects class.

    Kwargs:
        db (pymongo.database.Database): An instance representing the DB.

        meta (str): The path to the meta HDF5 file.

    Kwrvals:


    See Also:


    Notes:
        For subjects, the key '_id' is the preferred indexer.

    Example:


    References:
        .. [1]
```

**add**(self, subject=None, flag=True)

```
    To add a subject to the 'subjects' collection.

    Kwargs:
        subject (dict): Subject (JSON) to add.

        flag (bool): Flag to store in HDF5. Default: True.

    Kwrvals:
        subjectId (int): ID of the subject in the DB.

    See Also:


    Notes:
        If the subject already exists (i.e. there is a subject with the same name) the ID of the subject in the DB is returned.

    Example:
        subId = db.subjects.add({'name': 'subject'})['subjectId']

    References:
        .. [1]
```

**get**(self, refine={}, restrict={})

```
    Make a general query to the 'subjects' collection.

    Kwargs:
```

```
        refine (dict): Dictionary to refine the search. Default: {}.

        restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

    Kwrvals:
        docList (list): The list of dictionaries with the results of the query. Empty list if no match is found.

    See Also:


    Notes:


    Example:
        # get all subjects
        res = db.subjects.get()['docList']
        # get subjects with 'field' set to 'new'
        res = db.subjects.get(refine={'field': 'new'})['docList']
        # get subjects with 'field' set to 'new' and 'flag' set to False
        res = db.subjects.get(refine={'field': 'new', 'flag': False})['docList']

    References:
        .. [1]
```

**getById**(self, subjectId=None, restrict={})

```
    Query the 'subjects' collection by ID.

    Kwargs:
        subjectId (int): ID of the subject to query.

        restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

    Kwrvals:
        doc (dict): The document with the results of the query. None if no match is found.

    See Also:


    Notes:


    Example:
        # get everything
        doc = db.subjects.getById(0)['doc']
        # only return the ID
        doc = db.subjects.getById(0, restrict={'_id': 1})['doc']
        # only return the name (ID is also returned!)
        doc = db.subjects.getById(0, restrict={'name': 1})['doc']
        # only return the name and don't return the ID
        doc = db.subjects.getById(0, restrict={'name': 1, '_id': 0})['doc']

    References:
        .. [1]
```

**getByName**(self, subjectName=None, restrict={})

```
    Query the 'subjects' collection by name.

    Kwargs:
        subjectName (str): Name of the subject to query.

        restrict (dict): Dictionary to restrict the information sent by the DB. Default: {}.

    Kwrvals:
        doc (dict): The document with the results of the query. None if no match is found.

    See Also:


    Notes:


    Example:
        # get everything
        doc = db.subjects.getByName('subject')['doc']
        # only return the ID
        doc = db.subjects.getByName('subject', restrict={'_id': 1})['doc']
        # only return the name (ID is also returned!)
        doc = db.subjects.getByName('subject', restrict={'name': 1})['doc']
        # only return the name and don't return the ID
        doc = db.subjects.getByName('subject', restrict={'name': 1, '_id': 0})['doc']

    References:
        .. [1]
```

**list**(self)

```
    List all the subjects in the database.

    Kwargs:


    Kwrvals:
        idList (list): List with the IDs of the subjects in the database.

    See Also:


    Notes:
```

```
    Example:
        subList = db.subjects.list()['idList']

    References:
        .. [1]
```

**update**(self, subjectId=None, info={})

```
    Update a subject with the given information. Fields can be added, and its type changed, but not deleted.

    Kwargs:
        subjectId (int): ID of the subject to update.

        info (dict): Dictionary with the information to update. Default: {}.

    Kwrvals:


    See Also:


    Notes:


    Example:
        db.subjects.update(0, {'new': 'field'})

    References:
        .. [1]
```

## Functions

**listDB**(host='localhost', port=27017)

```
    Lists the databases present in a given server.

    Kwargs:
        host (str): Network address of the MongoDB server. Default: 'localhost'.

        port (int): Port the MongoDB server is listening on. Default: 27017.

    Kwrvals:
        dbList (list): A list with DB names.

    See Also:


    Notes:


    Example:
        listDB('193.136.222.234', 27017)

    References:
        .. [1]
```

## Data

**RETAG** = <_sre.SRE_Pattern object>

# bspyclone.database.philipsXML

index
d:\work\productioncode\clones\biosppy\bspyclone\database\philipsxml.py

```
.. module:: philipsXML
   :platform: Windows
   :synopsis: This module provides tools to read ECG records stored using the Philips SierraECG standard.

.. moduleauthor:: Carlos Carreiras
```

## Modules

lxml.etree
bspyclone.database.h5db
numpy
os
xmltodict

## Functions

**readXML**(fpath=None, validate=True)

```
Read a decompressed Philips XML file.

Kwargs:
    fpath (str): XML file to read.

    validate (bool): If True, validates the XML schema.

Kwrvals:
    date (str): ISO 8601 acquisition date and time.

    labels (list): Label for each acquired ECG channel.

    resolution (int): Signal resolution.

    sampleRate (float): Acquisition sample rate.

    signal (array): Signal array, where each line is an ECG channel.

    subjectID (str): The ID of the subject.

    subjectName (str): The name of the subject.

    xmlFileName (str): Name of the source XML file.

See Also:


Notes:


Example:


References:
    .. [1]
```

# bspyclone.database.physionet

index
d:\work\productioncode\clones\biosppy\bspyclone\database\physionet.py

```
.. module:: biomesh
   :platform: Unix, Windows
   :synopsis: Convert Physionet database files to csv (requires wfdb).

.. moduleauthor:: Carlos Carreiras
```

## Modules

csv
fnmatch
numpy
os
subprocess

## Functions

**batchConvert**(basePath, db, dirPath=None)

**convert2csv**(basePath, record, dirPath)

**convertAnnotation2CSV**(basePath, record, dirPath)

**listFiles**(basePath, db)

**readCSV**(path)

# bspyclone.ecg.ecg

index
d:\work\productioncode\clones\biosppy\bspyclone\ecg\ecg.py

```
.. module:: ecg
   :platform: Unix, Windows
   :synopsis: This module provides various functions to handle ECG signals.

.. moduleauthor:: Filipe Canento
```

## Modules

bspyclone.ecg.models
bspyclone.filt
bspyclone.plux
pylab
scipy
bspyclone.tools
unittest

## Functions

**ecg**(Signal=None, SamplingRate=1000.0, Filter={})

```
Determine ECG signal information.

Kwargs:
    Signal (array): input ECG signal.

    SamplingRate (float): Sampling frequency (Hz).

    Filter (dict): Filter parameters.

Kwrvals:
    R (array): heart beat indexes (or instants in seconds if sampling rate is defined).

Configurable fields:{"name": "ecg.ecg", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["R"]}

See Also:
    filt
    models.hamilton

Notes:

Example:

References:
    .. [1]
```

**features**(Signal=None, SamplingRate=1000.0, Filter={})

```
Retrieves relevant ECG signal features.

Kwargs:
    Signal (array): input ECG signal.

    SamplingRate (float): Sampling frequency (Hz).

    Filter (dict): Filter parameters.

Kwrvals:
    R (array): ECG R-peak indexes (or instants in seconds if sampling rate is defined)

    mean (float): mean

    std (float): standard deviation

    var (float): variance

    skew (ndarry): skewness

    kurtosis (array): kurtosis

    ad (float): absolute deviation

Configurable fields:
{"name": "ecg.features", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["R", "mean", "std", "var", "skew",

See Also:
    filt
    models.hamilton
    tls.statsf

Notes:

Example:

References:
    .. [1]
```

**filt**(Signal=None, SamplingRate=1000.0, UpperCutoff=16.0, LowerCutoff=8.0, Order=4)

```
Filters an input ECG signal.

By default, the return is the filtered Signal assuming a
1000Hz sampling frequency and the following filter sequence:
    1. 4th order low-pass filter with cutoff frequency of 16Hz;
    2. 4th order high-pass filter with cutoff frequency of 8Hz;
    3. d[]/dt;
```

```
        4. 80ms Hamming Window Smooth.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        UpperCutoff (float): Low-pass filter cutoff frequency (Hz).

        LowerCutoff (float): High-pass filter cutoff frequency (Hz).

        Order (int): Filter order.

    Kwrvals:
        Signal (array): output filtered signal.

    Configurable fields:
    {"name": "ecg.filt", "config": {"UpperCutoff": "16.", "SamplingRate": "1000.", "LowerCutoff": "8.", "Order": "4"}, "inputs": ["Signal"], "out

    See Also:
        flt.zpdfr

        flt.smooth

    Notes:

    Example:
        Signal = load(...)

        SamplingRate = ...

        res = filt(Signal=Signal, SamplingRate=SamplingRate)

        plot(res['Signal'])

    References:
        .. [1] P.S. Hamilton, Open Source ECG Analysis Software Documentation, E.P.Limited
        http://www.eplimited.com/osea13.pdf
```

# bspyclone.ecg.hrv

index
d:\work\productioncode\clones\biosppy\bspyclone\ecg\hrv.py

```
.. module:: hrv
   :platform: Unix, Windows
   :synopsis: This module provides various methods to perform Heart Rate Variability analysis.

.. moduleauthor:: Carlos Carreiras
```

## Modules

numpy  scipy.signal

## Functions

**hrv**(R, sampleRate, npoints=100000)

```
    Compute Heart Rate Variability (HRV) metrics from a sequence of R peak locations.

    Kwargs:
        R (list, array): Positions of the R peaks in samples (from a segmentation algorithm).

        sampleRate (float): Sampling rate (Hz).

        npoints (int): Number of frequency points for the Lomb-
    Scargle periodogram. Default=100000.

    Kwrvals:
        time (array): Instantaneous heart rate time points.

        HR (array): Instantaneous heart rate.

        RR (array): Instantaneous RR intervals.

        mHR (float): Mean heart rate.

        SD (float): Standard deviation of heart rate.

        RMSSD (float): Root mean square of successive differences.

        HF (float): High frequency power (0.15 to 0.4 Hz).

        LF (float): Low frequency power (0.04 to 0.15 Hz).

        L2HF (float): Ratio of LF to HF power.

    See Also:


    Notes:
        Maybe deal with un/mis-detected R peaks

    Example:


    References:
        .. [1]
```

# bspyclone.ecg.models

index
d:\work\productioncode\clones\biosppy\bspyclone\ecg\models.py

```
.. module:: models
   :platform: Unix, Windows
   :synopsis: This module provides various methods to segment ECG signals.

.. moduleauthor:: Filipe Canento, Carlos Carreiras, Francisco David
```

## Modules

collections
numpy
pylab
scipy
scipy.signal
bspyclone.ecg.tools
traceback

## Functions

**ESSF**(Signal=None, SamplingRate=1000.0, ESSF_params={'s_amp': 70.0, 's_win': 0.3})

```
ECG Slope Sum Function: algorithm to detect ECG beat indexes.

Kwargs:
    Signal (array): input filtered ECG signal.

    SamplingRate (float): Sampling frequency (Hz).

    ESSF_params (float): data dependent parameters.

Returns:
    rpeaks (array): R-peak indexes

Configurable fields:{"name": "models.ESSF", "config": {"SamplingRate": "1000."}, "inputs": ["Signal"], "outputs": ["rpeaks"]}

See Also:

Notes:

Example:

References:
    .. [1] ...
```

**armSSF**(Signal=None, SamplingRate=1000.0, threshold=20, winB=0.03, winA=0.01, Params=None)

```
Modified Slope Sum Function R peak detection algorithm for ARM board.

Kwargs:
    Signal (array): input ECG signal.

    SamplingRate (float): Sampling frequency (Hz).

    threshold (int): threshold.

    winB (int): size of search window before candidate.

    winA (int): size of search window after candidate.

Returns:
    rpeaks (array): R-peak indexes.

Configurable fields:{}

See Also:

Notes:

Example:

References:
    .. [1] ...
```

**batch_christov**(Signal=None, SamplingRate=1000.0, debug=False, IF=True)

**batch_engzee**(Signal=None, SamplingRate=1000.0, debug=False, IF=True)

```
Kwargs:


Kwrvals:


Configurable fields:{"name": "models.batch_engzee", "config": {"SamplingRate": "1000.0"}, "inputs": ["Signal"], "outputs": []}

See Also:


Notes:


Example:


References:
    .. [1]
```

**christov**(Signal=None, SamplingRate=1000.0, Filter={}, Params={}, Show=False)

```
Determine ECG signal information.

Kwargs:
    Signal (array): input ECG signal.

    SamplingRate (float): Sampling frequency (Hz).

    Filter (dict): Filter parameters.

    Params (dict): Initial conditions.

Kwrvals:

Configurable fields:{"name": "models.christov", "config": {"SamplingRate": "1000.", "Show": "False"}, "inputs": ["Signal", "Filter", "Params"

See Also:

Notes:

Example:

References:
    .. [1]    Ivaylo I Christov, Real time electrocardiogram QRS detection using combined adaptive threshold,
            BioMedical Engineering OnLine 2004, 3:28
            This article is available from: http://www.biomedical-engineering-online.com/content/3/1/28
            2004 Christov; licensee BioMed Central Ltd.
```

**definepeak**(signal, srate)

**engzee**(Signal=None, SamplingRate=1000.0, Params=None)

```
Determine ECG signal information. Adaptation of the Engelse and Zeelenberg by [1].

Kwargs:
    Signal (array): input ECG signal.

    SamplingRate (float): Sampling frequency (Hz).

    Params (dict): Initial conditions:
        CSignal (array): Continuity signal

        MM (array): MM threshold

        MMidx (int): MM threshold current index

        NN (array):  NN threshold

        NNidx (int): NN threshold current index

        offset (array): last intersection points

        prevR (collections.deque): Last R positions

        Rminus (float): Segmentation window size to the left of R (default: 200 ms)

        Rplus (float): Segmentation window size to the right of R (default: 400 ms)

        update (bool): Update flag

Kwrvals:
    R (array): R peak indexes

    Segments (array): Extracted segments

    HH (float): Heart rate

    Params (dict):
        CSignal (array): Continuity signal

        MM (array): MM threshold

        MMidx (int): MM threshold current index

        NN (array):  NN threshold

        NNidx (int): NN threshold current index

        offset (array): last intersection points

        prevR (collections.deque): Last R positions

        Rminus (float): Segmentation window size to the left of R (default: 200 ms)

        Rplus (float): Segmentation window size to the right of R (default: 400 ms)

        update (bool): Update flag

See Also:


Notes:


Example:


References:
```

```
        .. [1] Andre Lourenco, Hugo Silva, Paulo Leite, Renato Lourenco and Ana Fred,
            REAL TIME ELECTROCARDIOGRAM SEGMENTATION FOR FINGER BASED ECG BIOMETRICS
```

**engzee_incomplete**(Signal=None, SamplingRate=1000.0, Filter=None, Params=None, initialFilter=False)

```
    --- REVIEW DOCS ---

    Determine ECG signal information. Adaptation of the Engelse and Zeelenberg by [1].

    Kwargs:
        Signal (array): input ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): Filter parameters.

        Params (dict): Initial conditions

            Segments (list): list of ECG segments

            RawSegments (list): list of Raw ECG segments

            MM (array): MM threshold

            MMidx (int): MM threshold current index

            offset (array): last intersection points

            offsetidx (int): offset current index

            rpeak (array): R indexes

            nthfpluss (array): nthfplus intersection points

            Rminus (int): Segmentation window size to the left of R (default: 200 ms)

            Rplus (int): Segmentation window size to the right of R (default: 400 ms)

        initialFilter (bool): Apply hands ECG filter (default: False).


    Kwrvals:
        R (array): R peak indexes

        Params (dict):

            Segments (list): list of ECG segments

            RawSegments (list): list of Raw ECG segments

            MM (array): MM threshold

            MMidx (int): MM threshold current index

            offset (array): last intersection points

            offsetidx (int): offset current index

            rpeak (array): R indexes

            nthfpluss (array): nthfplus intersection points

            Rminus (int): Segmentation window size to the left of R (default: 200 ms)

            Rplus (int): Segmentation window size to the right of R (default: 400 ms)

    Configurable fields:
    {"name": "models.engzee", "config": {"SamplingRate": "1000.", "initialfilter": "True"}, "inputs": ["Signal", "Filter", "Params"], "outputs":

    See Also:


    Notes:
        1. Tested for chest ECGs: paper thresholds, no initial filtering

        2. Finger/palm ECGs: there's an initial filtering and some parameters were modified

    Example:
        # Data filename

        fname = ...

        # Load data

        dataecg = plux.loadbpf(fname, usecols=(3,))

        # Init

        mrkr = 0

        SamplingRate = dataecg.header['SamplingFrequency']

        win = 3*SamplingRate

        mrkrend = win

        # ECG algorithm, segmentation: R-200ms to R+400ms
```

```
        res = ecgmodule.models.engzee(Signal=dataecg[:win], SamplingRate=SamplingRate, Params={'Rminus': int(0.2*SamplingRate), 'Rplus': int(0.4

        dlen = len(dataecg)

        while ( mrkrend < dlen ):

            mrkr = res['Params']['offset'][-1]

            res = ecgmodule.models.engzee(Signal=dataecg[mrkr:mrkrend], SamplingRate=SamplingRate, Params=res['Params'])

        ECGSegments = res['Params']['Segments']

    References:
        .. [1] Andre Lourenco, Hugo Silva, Paulo Leite, Renato Lourenco and Ana Fred,
        REAL TIME ELECTROCARDIOGRAM SEGMENTATION FOR FINGER BASED ECG BIOMETRICS
```

**engzee_old**(Signal=None, SamplingRate=1000.0, initialfilter=True, Filter={}, Params={})

```
    ### HAS PROBLEM WITH R PEAKS LOCATION IN ONLINE USE-CASE

    Determine ECG signal information. Adaptation of the Engelse and Zeelenberg by [1].

    Kwargs:
        Signal (array): input ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): Filter parameters.

        Params (dict): Initial conditions

            Segments (list): list of ECG segments

            RawSegments (list): list of Raw ECG segments

            MM (array): MM threshold

            MMidx (int): MM threshold current index

            offset (array): last intersection points

            offsetidx (int): offset current index

            rpeak (array): R indexes

            nthfpluss (array): nthfplus intersection points

            Rminus (int): Segmentation window size to the left of R (default: 200 ms)

            Rplus (int): Segmentation window size to the right of R (default: 400 ms)


    Kwrvals:
        R (array): R peak indexes

        Params (dict):

            Segments (list): list of ECG segments

            RawSegments (list): list of Raw ECG segments

            MM (array): MM threshold

            MMidx (int): MM threshold current index

            offset (array): last intersection points

            offsetidx (int): offset current index

            rpeak (array): R indexes

            nthfpluss (array): nthfplus intersection points

            Rminus (int): Segmentation window size to the left of R (default: 200 ms)

            Rplus (int): Segmentation window size to the right of R (default: 400 ms)

    Configurable fields:
    {"name": "models.engzee", "config": {"SamplingRate": "1000.", "initialfilter": "True"}, "inputs": ["Signal", "Filter", "Params"], "outputs":

    See Also:


    Notes:
        1. Tested for chest ECGs: paper thresholds, no initial filtering

        2. Finger/palm ECGs: there's an initial filtering and some parameters were modified

    Example:
        # Data filename

        fname = ...

        # Load data

        dataecg = plux.loadbpf(fname, usecols=(3,))

        # Init
```

```
        mrkr = 0

        SamplingRate = dataecg.header['SamplingFrequency']

        win = 3*SamplingRate

        mrkrend = win

        # ECG algorithm, segmentation: R-200ms to R+400ms

        res = ecgmodule.models.engzee(Signal=dataecg[:win], SamplingRate=SamplingRate, Params={'Rminus': int(0.2*SamplingRate), 'Rplus': int(0.4

        dlen = len(dataecg)

        while ( mrkrend < dlen ):

            mrkr = res['Params']['offset'][-1]

            res = ecgmodule.models.engzee(Signal=dataecg[mrkr:mrkrend], SamplingRate=SamplingRate, Params=res['Params'])

        ECGSegments = res['Params']['Segments']

    References:
        .. [1] Andre Lourenco, Hugo Silva, Paulo Leite, Renato Lourenco and Ana Fred,
        REAL TIME ELECTROCARDIOGRAM SEGMENTATION FOR FINGER BASED ECG BIOMETRICS
```

**gamboa**(Signal=None, SamplingRate=1000.0, tol=0.002)

```
    Gamboa's algorithm to detect ECG beat indexes.

    Kwargs:
        Signal (array): input filtered ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        tol (float): tolerance parameter.

    Returns:
        rpeaks (array): R-peak indexes

    Configurable fields:{"name": "models.gamboa", "config": {"SamplingRate": "1000."}, "inputs": ["Signal"], "outputs": ["rpeaks"]}

    See Also:

    Notes:

    Example:

    References:
        .. [1] ...
```

**hamilton**(Signal=None, SamplingRate=1000.0, Params=None, hand=True)

```
    Algorithm to detect ECG beat indexes.

    Kwargs:
        Signal (array): input filtered ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        Params (dict): Initial conditions:

        hand (bool): Signal is obtained from the hands.

    Kwrvals:
        Signal (array): output filtered signal if Filter is defined.

        R (array): R peak indexes (or instants in seconds if sampling rate is defined).

        init (dict): dict with initial values of some variables

            npeaks (int): number of detected heart beats.

            indexqrs (int): most recent QRS complex index.

            indexnoise (int): most recent noise peak index.

            indexrr (int): most recent R-to-R interval index.

            qrspeakbuffer (array): 8 most recent QRS complexes.

            noisepeakbuffer (array): 8 most recent noise peaks.

            rrinterval (array): 8 most recent R-to-R intervals.

            DT (float): QRS complex detection threshold.

            offset (int): signal start in samples.

    Configurable fields:
    {"name": "models.hamilton", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter", "init"], "outputs": ["Signal", "R", "init",

    See Also:
        filt

    Notes:

    Example:
```

```
    References:
        .. [1] P.S. Hamilton, Open Source ECG Analysis Software Documentation, E.P.Limited
        http://www.eplimited.com/osea13.pdf
```

**hamilton_old**(Signal=None, SamplingRate=1000.0, Filter=(), init=())

```
    Algorithm to detect ECG beat indexes.

    Kwargs:
        Signal (array): input filtered ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): Filter parameters.

    Kwrvals:
        Signal (array): output filtered signal if Filter is defined.

        R (array): R peak indexes (or instants in seconds if sampling rate is defined).

        init (dict): dict with initial values of some variables

            npeaks (int): number of detected heart beats.

            indexqrs (int): most recent QRS complex index.

            indexnoise (int): most recent noise peak index.

            indexrr (int): most recent R-to-R interval index.

            qrspeakbuffer (array): 8 most recent QRS complexes.

            noisepeakbuffer (array): 8 most recent noise peaks.

            rrinterval (array): 8 most recent R-to-R intervals.

            DT (float): QRS complex detection threshold.

            offset (int): signal start in samples.

    Configurable fields:
    {"name": "models.hamilton", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter", "init"], "outputs": ["Signal", "R", "init", "

    See Also:
        filt

    Notes:

    Example:

    References:
        .. [1] P.S. Hamilton, Open Source ECG Analysis Software Documentation, E.P.Limited
        http://www.eplimited.com/osea13.pdf
```

**hamilton_tst**(Signal=None, SamplingRate=1000.0, init=None, hand=True)

```
    Algorithm to detect ECG beat indexes.

    Kwargs:
        Signal (array): input filtered ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): Filter parameters.

    Kwrvals:
        Signal (array): output filtered signal if Filter is defined.

        R (array): R peak indexes (or instants in seconds if sampling rate is defined).

        init (dict): dict with initial values of some variables

            npeaks (int): number of detected heart beats.

            indexqrs (int): most recent QRS complex index.

            indexnoise (int): most recent noise peak index.

            indexrr (int): most recent R-to-R interval index.

            qrspeakbuffer (array): 8 most recent QRS complexes.

            noisepeakbuffer (array): 8 most recent noise peaks.

            rrinterval (array): 8 most recent R-to-R intervals.

            DT (float): QRS complex detection threshold.

            offset (int): signal start in samples.

    Configurable fields:
    {"name": "models.hamilton", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter", "init"], "outputs": ["Signal", "R", "init", "

    See Also:
        filt

    Notes:
```

```
    Example:

    References:
        .. [1] P.S. Hamilton, Open Source ECG Analysis Software Documentation, E.P.Limited
        http://www.eplimited.com/osea13.pdf
```

**happee**(Signal=None, SamplingRate=1000.0, window=3.0, overlap=0.75, threshold=0.0, gridMem=None)

```
    Implementation of the Highly Accurate Pulse Predictor for Exercise Equipment (HAPPEE) [1].

    Kwargs:
        Signal (array): input ECG signal.

        SamplingRate (float): Sampling frequency (Hz).

        window (float): Size (seconds) of the search window.

        overlap (float): Percentage of window overlap.

        threshold (float): Pulse detection threshold.

        gridMem (dict): helper grid dictionary.

    Returns:


    Configurable fields:{}

    See Also:

    Notes:

    Example:

    References:
        .. [1] Kay, S.; Ding, Q.; Li, D., "On the Design and Implementation of A Highly Accurate
               Pulse Predictor for Exercise Equipment," IEEE Trans. on Biomedical Engineering,
               10.1109/TBME.2015.2407155
```

**happee_grid**(length, grid, M)

```
    Generate the helper grid dictionary for the HAPPEE algorithm [1].

    Kwargs:
        length (int): size of the signal window.

        grid (array, list): search grid of pulse periods.

        M (int): Pulse length.

    Returns:
        gridMem (dict): helper grid dictionary.


    Configurable fields:{}

    See Also:

    Notes:

    Example:

    References:
        .. [1] Kay, S.; Ding, Q.; Li, D., "On the Design and Implementation of A Highly Accurate
               Pulse Predictor for Exercise Equipment," IEEE Trans. on Biomedical Engineering,
               10.1109/TBME.2015.2407155
```

**happee_search**(window, grid, M, gridMem)

```
    Perform HAPPEE [1] search on a signal window.

    Kwargs:
        window (array): input signal window.

        grid (array, list): search grid of pulse periods.

        M (int): Pulse length.

        gridMem (dict): helper grid dictionary.

    Returns:
        T (float): Test statistic.

        n0 (int): Location of first pulse.

        P (int): Pulse period.


    Configurable fields:{}

    See Also:

    Notes:

    Example:

    References:
        .. [1] Kay, S.; Ding, Q.; Li, D., "On the Design and Implementation of A Highly Accurate
               Pulse Predictor for Exercise Equipment," IEEE Trans. on Biomedical Engineering,
               10.1109/TBME.2015.2407155
```

**happee_variance**(signal)

```
Compute HAPPEE variance [1].

Kwargs:
    signal (array): input signal.

Returns:
    variance (float): HAPPEE variance.

Configurable fields:{}

See Also:

Notes:

Example:

References:
    .. [1] Kay, S.; Ding, Q.; Li, D., "On the Design and Implementation of A Highly Accurate
           Pulse Predictor for Exercise Equipment," IEEE Trans. on Biomedical Engineering,
           10.1109/TBME.2015.2407155
```

**monhe**(Signal=None, SamplingRate=1000.0)

```
GREAT but:
    discard crossings close to one another by less than 100 ms
```

**monhe2**(Signal=None, Filtered=None, SamplingRate=1000.0, checkSign=True)

```
GREAT but:
    discard crossings close to one another by less than 100 ms
```

# bspyclone.ecg.tools

index

d:\work\productioncode\clones\biosppy\bspyclone\ecg\tools.py

Created on May 28, 2013

@author: Carlos

## Modules

numpy pylab scipy.signal

## Functions

**PLFCorrSynchronize**(a, v, flag=False)

**PLFSynchronize**(a, v, flag=False, minOverlap=1)

**checkECG**(data)

**compareSegmentation**(referenceR=None, testR=None, SamplingRate=None, offset=0, minRR=None, tol=0.05)

**extractHeartbeats**(signal, R, sampleRate, before=0.2, after=0.4)

**getSyncSlices**(d, a, v)

**syncOverlap**(a, v)

**synchronize**(a, v, flag=False)

# bspyclone.eda.eda

index
d:\work\productioncode\clones\biosppy\bspyclone\eda\eda.py

```
.. module:: eda
   :platform: Unix, Windows
   :synopsis: This module provides various functions to handle EDA signals.

.. moduleauthor:: Filipe Canento
```

## Modules

bspyclone.eda.models
numpy
pylab
sys
unittest

## Functions

**features**(Signal=None, SamplingRate=1000.0, Filter={})

```
    Retrieves relevant EDA signal features.

    Kwargs:
        Signal (array): input EDA signal

        SamplingRate (float): sampling frequency (Hz)

        Filter (dict): Filter parameters

    Kwrvals:
        Signal (array): output filtered signal (see notes 1)

        Amplitude (array): signal pulses amplitudes (in the units of the input signal)

        Onset (array): indexes (or instants in seconds, see notes 2.a) of the SCRs onsets

        Peak (array): indexes (or instants in seconds, see notes 2.a) of the SCRs peaks

        Rise (array): SCRs rise times (in seconds)

        HalfRecovery (array): SCRs half-recovery times (in seconds)

        mean (float): mean

        std (float): standard deviation

        var (float): variance

        skew (ndarry): skewness

        kurtosis (array): kurtosis

        ad (float): absolute deviation

    Configurable fields:
    {"name": "eda.features", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "Amplitude", "Onset", "Pe

    See Also:
        filt
        models.basicSCR
        tls.statsf

    Notes:
        1 - If a filter is given as a parameter, then the returned keyworded values dict has a 'Signal' key.

        2 - If the sampling rate is defined, then:
            a) keys 'onset', and 'peak' are converted to instants of occurrence in seconds.

    Example:


    References:
        .. [1]
```

**filt**(Signal=None, SamplingRate=1000.0, UpperCutoff=0.25, Order=2)

```
    Filters an input EDA signal.

    If only input signal is provide, it returns the filtered EDA signal
    assuming a 1000Hz sampling frequency and a default low-pass filter
    with a cutoff frequency of 0.25Hz.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        UpperCutoff (float): Low-pass filter cutoff frequency (Hz).

        LowerCutoff (float): High-pass filter cutoff frequency (Hz).

        Order (int): Filter order.

    Kwrvals:
        Signal (array): output filtered signal.
```

```
    Configurable fields:{"name": "eda.filt", "config": {"UpperCutoff": "0.25", "SamplingRate": "1000.", "Order": "2"}, "inputs": ["Signal"], "out

    See Also:
        flt.zpdfr

    Notes:


    Example:
        Signal = load(...)

        SamplingRate = ...

        res = filt(Signal=Signal, SamplingRate=SamplingRate)

        plot(res['Signal'])

    References:
        .. [1]
```

**scl**(Signal=None, SamplingRate=1000.0, Filter={'UpperCutoff': 0.05})

```
    Kwargs:


    Kwrvals:


    Configurable fields:{"name": "eda.scl", "config": {"Filter": "{"UpperCutoff":0.05}", "SamplingRate": "1000."}, "inputs": ["Signal"], "outputs

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**scr**(Signal=None, SamplingRate=1000.0, Method='basic', Filter={})

```
    Detects and extracts Skin Conductivity Responses (SCRs) information such as:
    SCRs amplitudes, onsets, peak instant, rise, and half-recovery times.

    Kwargs:
        Signal (array): input EDA signal.

        SamplingRate (float): Sampling frequency (Hz).

        Method (string): SCR detection algorithm.

        Filter (dict): filter parameters.

    Kwrvals:
        Signal (array): output filtered signal (see notes 1)

        Amplitude (array): signal pulses amplitudes (in the units of the input signal)

        Onset (array): indexes (or instants in seconds, see notes 2.a) of the SCRs onsets

        Peak (array): indexes (or instants in seconds, see notes 2.a) of the SCRs peaks

        Rise (array): SCRs rise times (in seconds)

        HalfRecovery (array): SCRs half-recovery times (in seconds)

    Configurable fields:{"name": "eda.scr", "config": {"SamplingRate": "1000.", "Method": ""basic""}, "inputs": ["Signal", "Filter"], "outputs":

    See Also:
        filt

        models.basicSCR

        models.KBKSCR

    Notes:
        1 - If a filter is given as a parameter, then the returned keyworded values dict has a 'Signal' key.

        2 - If the sampling rate is defined, then:
            a) keys 'onset', and 'peak' are converted to instants of occurrence in seconds.

    Example:


    References:
        .. [1]
```

# bspyclone.eda.models

index
d:\work\productioncode\clones\biosppy\bspyclone\eda\models.py

## Modules

numpy pylab sys unittest

## Functions

**KBKSCR**(Signal=None, SamplingRate=1000.0)

```
    Detects and extracts Skin Conductivity Responses (SCRs) information such as:
    SCRs amplitudes, onsets, peak instant, rise, and half-recovery times.

    Kwargs:
            Signal (array): input EDA signal.

            SamplingRate (float): Sampling frequency (Hz).

    Kwrvals:
            Signal (array): output filtered signal (see notes 1)

            Amplitude (array): signal pulses amplitudes (in the units of the input signal)

            Onset (array): indexes (or instants in seconds, see notes 2.a) of the SCRs onsets

            Peak (array): indexes (or instants in seconds, see notes 2.a) of the SCRs peaks

            TODO: Rise (array): SCRs rise times (in seconds)

            TODO: HalfRecovery (array): SCRs half-recovery times (in seconds)

    See Also:
            flt.zpdfr

    Notes:
            1 - If the sampling rate is defined, then:
                    a) keys 'onset', and 'peak' are converted to instants of occurrence in seconds.

            2- Less sensitive than Gamboa algorithm, but does not solve the overlapping SCRs problem.

    Example:


    References:
            .. [1]  K.H. Kim, S.W. Bang, and S.R. Kim
                            "Emotion recognition system using short-
    term monitoring of physiological signals"
                            Med. Biol. Eng. Comput., 2004, 42, 419-427
```

**basicSCR**(Signal=None, SamplingRate=1000.0, Filter={})

```
    Detects and extracts Skin Conductivity Responses (SCRs) information such as:
    SCRs amplitudes, onsets, peak instant, rise, and half-recovery times.

    Kwargs:
        Signal (array): input EDA signal.

        SamplingRate (float): Sampling frequency (Hz).

        Method (string): SCR detection algorithm.

        Filter (dict): filter parameters.

    Kwrvals:
        Signal (array): output filtered signal (see notes 1)

        Amplitude (array): signal pulses amplitudes (in the units of the input signal)

        Onset (array): indexes (or instants in seconds, see notes 2.a) of the SCRs onsets
```

```
        Peak (array): indexes (or instants in seconds, see notes 2.a) of the SCRs peaks

        Rise (array): SCRs rise times (in seconds)

        HalfRecovery (array): SCRs half-recovery times (in seconds)

    See Also:
        filt

    Notes:
        1 - If a filter is given as a parameter, then the returned keyworded values dict has a 'Signal' key.

        2 - If the sampling rate is defined, then:
            a) keys 'onset', and 'peak' are converted to instants of occurrence in seconds.

    Example:


    References:
        .. [1]
```

**gamboa**(to, tp, yo, yp)

```
    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
            .. [1]
```

# bspyclone.eeg.eeg



EEG functions

## Modules

scipy.interpolate
numpy
pandas
pylab
scipy.signal
scipy.stats.stats

## Functions

**EMDiit**(Signal=None, niter=20, coef=0.6)

**HSPlot**(T, W, matrix, NTicks=8)

**HilbertSpectrum**(IMF, Fs, NF=100)

**ICAFilter**(signal=None)

**analyticSignal**(signal=None, axis=-1)

**bandPower**(signal=None, Fs=1, lower=0, upper=1, NFFT=4096, axis=-1)

**car**(signal=None)

**emd**(data, extrapolation='mirror', nimfs=12, sifting_distance=0.2)

```
Perform a Empirical Mode Decomposition on a data set.

This function will return an array of all the Emperical Mode Functions as
defined in [1]_, which can be used for further Hilbert Spectral Analysis.

The EMD uses a spline interpolation function to approximate the upper and
lower envelopes of the signal, this routine implements a extrapolation
routine as described in [2]_ as well as the standard spline routine.
The extrapolation method removes the artifacts introduced by the spline fit
at the ends of the data set, by making the dataset a continuious circle.

    Reproduced from github.com/jaidevd/pyhht

Parameters
----------
data : array_like
        Signal Data
extrapolation : str, optional
        Sets the extrapolation method for edge effects.
        Options: None
                 'mirror'
        Default: 'mirror'
nimfs : int, optional
        Sets the maximum number of IMFs to be found
        Default : 12
sifiting_distance : float, optional
        Sets the minimum variance between IMF iterations.
        Default : 0.2

Returns
-------
IMFs : ndarray
        An array of shape (len(data),N) where N is the number of found IMFs

Notes
-----

References
----------
.. [1] Huang H. et al. 1998 'The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-
stationary time series analysis.'
Procedings of the Royal Society 454, 903-995

.. [2] Zhao J., Huang D. 2001 'Mirror extending and circular spline function for empirical mode decomposition method'
Journal of Zhejiang University (Science) V.2, No.3,P247-252

.. [3] Rato R.T., Ortigueira M.D., Batista A.G 2008 'On the HHT, its problems, and some solutions.'
Mechanical Systems and Signal Processing 22 1374-1394
```

**intervalThresholding**(IMF=None, thr=None)

**laplacian**(signal=None, indList=None)

**noiseMeter**(data=None, Fs=None, norm=False)

**plf**(signal=None, pairList=None)

**power**(signal=None, NFFT=4096, axis=-1)

**spectPlot**(X, Y, Z, **kwarg)

**syncLikelihood**(signal=None)

**windower**(signal=None, Fs=None, length=128, shift=64, fcn=None, fcnArgs={})

# bspyclone.emg.emg

index
d:\work\productioncode\clones\biosppy\bspyclone\emg\emg.py

```
.. module:: emg
   :platform: Unix, Windows
   :synopsis: This module provides various functions to handle EMG signals.

.. moduleauthor:: Filipe Canento
```

## Modules

numpy
pylab
scipy
sys
unittest

## Functions

**emg**(Signal=None, SamplingRate=1000.0, Filter=())

```
    EMG signal processing and feature extraction.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): filter parameters.

    Kwrvals:
        Signal (array): output filtered signal (see notes 1).

        onoff (array): indexes of EMG onsets

    Configurable fields:{"name": "emg.emg", "config": {"SamplingRate": "1000.0"}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "onoff"]}

    See Also:
        filt

        onoff

    Notes:

    Example:


    References:
        .. [1]
```

**features**(Signal=None, SamplingRate=1000.0, Filter={})

```
    Retrieves relevant EMG signal features.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): filter parameters.

    Kwrvals:
        Signal (array): output filtered signal (see notes 1).

        onoff (array): indexes of EMG onsets

        mean (float): mean

        std (float): standard deviation

        var (float): variance

        skew (ndarry): skewness

        kurtosis (array): kurtosis

        ad (float): absolute deviation

    Configurable fields:
    {"name": "emg.features", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "onoff", "mean", "std", "

    See Also:
        filt

        onoff

        tls.statsf

    Notes:


    Example:


    References:
        .. [1]
```

**filt**(Signal=None, SamplingRate=1000.0, UpperCutoff=None, LowerCutoff=100.0, Order=4.0)

```
    Filters an input EMG signal.

    If only input signal is provide, it returns the filtered EMG signal
    assuming a 1000Hz sampling frequency and a default high-pass filter
    with a cutoff frequency of 100Hz.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        UpperCutoff (float): Low-pass filter cutoff frequency (Hz).

        LowerCutoff (float): High-pass filter cutoff frequency (Hz).

        Order (int): Filter order.

    Kwrvals:
        Signal (array): output filtered signal.

    Configurable fields:
    {"name": "emg.filt", "config": {"SamplingRate": "1000.", "LowerCutoff": "100.", "Order": "4."}, "inputs": ["Signal", "UpperCutoff"], "outputs

    See Also:
        flt.zpdfr

    Notes:


    Example:
        Signal = load(...)
        SamplingRate = ...
        res = filt(Signal=Signal, SamplingRate=SamplingRate)
        plot(res['Signal'])

    References:
        .. [1]
```

**onoff**(Signal=None, SamplingRate=1000.0, Thres=None, ws=50.0)

```
    EMG signal onset detection.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        Thres (float): detection threshold

        ws (float): detection window size in milliseconds

    Kwrvals:
        onoff (array): indexes of EMG onsets

    Configurable fields:{"name": "emg.onoff", "config": {"SamplingRate": "1000.", "ws": "50.0"}, "inputs": ["Signal", "Thres"], "outputs": ["onof

    See Also:
        flt.smooth

    Notes:

    Example:


    References:
        .. [1]
```

# bspyclone.filt

index
d:\work\productioncode\clones\biosppy\bspyclone\filt.py

## Modules

numpy
pylab
scipy.signal
unittest
scipy.signal.windows

## Functions

**filterSignal**(Signal=None, SamplingRate=1000.0, FilterType='FIR', Order=None, Frequency=None, BandType='lowpass', **kwargs)

```
Filter a signal according to the given parameters.
Uses a forward-backward filter implementation. Therefore, the combined filter has linear phase.

Supported filter functions: FIR, Butterworth, Chebyshev Type I, Chebyshev Type II, Elliptic, and Bessel.

Kwargs:
    Signal (array): The signal to filter.

    SamplingRate (int, float): The sampling frequency (Hz).

    FilterType (str): The filter function: 'FIR', 'butter', 'cheby1', 'cheby2', 'ellip', or 'bessel' (default='FIR').

    Order (int): Order of the filter.

    Frequency (int, float, list, array): The cutoff frequency (or list/array of low and high cutoff frequencies).

    BandType (str): The type of the filter: 'lowpass', 'highpass', 'bandpass', or 'bandstop' (default='lowpass').

    **kwargs (dict): Additional keyword arguments are passed to the underlying scipy.signal function.

Kwrvals:
    Signal (array): The filtered signal.

    SamplingRate (float): The sampling frequency (Hz).

    Filter (dict): The filter parameters.

See Also:
    _getFilter
    plotFilter
    scipy.signal.filtfilt

Notes:


Example:


References:
    .. [1]
```

**firfilt**(data, n, l, h, SamplingRate)

```
--DEPRECATED--
```

**plotFilter**(FilterType='FIR', Order=None, Frequency=None, SamplingRate=1000.0, BandType='lowpass', path=None, show=True, **kwargs)

```
Plot the frequency response of the filter specified with the given parameters.

Supported filter functions: FIR, Butterworth, Chebyshev Type I, Chebyshev Type II, Elliptic, and Bessel.

Kwargs:
    FilterType (str): The filter function: 'FIR', 'butter', 'cheby1', 'cheby2', 'ellip', or 'bessel' (default='FIR').

    Order (int): Order of the filter.

    Frequency (int, float, list, array): The cutoff frequency (or array of frequencies).

    SamplingRate (int, float): The sampling frequency (Hz).

    BandType (str): The type of the filter: 'lowpass', 'highpass', 'bandpass', or 'bandstop' (default='lowpass').

    path (str): If given, the plot will be saved to the file specified (default=None).

    show (bool): If True, show the plot immediately.
```

```
        **kwargs (dict): Additional keyword arguments are passed to the underlying scipy.signal function.

    Kwrvals:


    See Also:
        filterSignal

    Notes:


    Example:


    References:
        .. [1]
```

**smooth**(Signal=None, Window={})

```
    Smooth data using a N-point moving average filter.

    This implementation uses the convolution of a filter kernel with the input
    signal `x` to compute the smoothed signal.

    Parameters

    x : ndarray
    Input signal data.
    n : int, float, ndarray, optional
    Number of points of the filter kernel. If this is an `int`, the filter
    kernel will have `n` points. If this is a `float`, the number of points
    for the filter kernel will be set as `n*size(x)`. If this is a `ndarray`,
    it will be directly taken as the filter kernel.
    Default: 10
    wtype : str, function, optional
    Method that should be used to determine the filter kernel. If this is a
    `function`, it will be invoked with parameters `n`and `args` to determine
    the filter kernel. If this is a `str`, the function with the matching or
    most similar name belonging to the module ``scipy.signal.windows`` is used.
    Default: ``boxzen``
    *args : optional
    Additional parameters that may be required by the filter kernel function

    Returns

    y : ndarray
    A smoothed version of the input signal computed using a filter kernel of
    size `n` generated according to `wtype`

    See Also

    scipy.signal.windows

    Notes

    A combination smoothing method ``boxzen`` was introduced and is currently used
    as default to produce the output signal.

    This method first smooths the signal using the ``scipy.windows.boxcar`` window
    and then smooths ir again using the ``scipy.windows.parzen`` window.

    The resulting signals can be quite interesting, as ``boxcar``retains a great
    proximity to the original data waveforms, and ``parzen`` removes the rough edges.

    Example

    t = arange(0,2*pi,.1)
    x = sin(t)+0.5*(rand(len(t))-0.5)
    y = smooth(x)
    plot(x,'.')
    plot(y)
    legend(('original data', 'smoothed with boxzen'))

    References

    .. [1] Wikipedia, "Moving Average". http://en.wikipedia.org/wiki/Moving_average
    .. [2] S. W. Smith, "Moving Average Filters - Implementation by Convolution".
    http://www.dspguide.com/ch15/1.htm

    Kwargs:


    Kwrvals:
```

```
    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**windowfcn**(w)

```
    Retrieve the appropriate window function that corresponds to the descriptor `w`.

    Parameters

    w : str, function
    Name or descriptor of the window function. If this is a `function`, it will be
    directly returned. If this is a `str`, a window function with a matching name
    will be searched for in module ``scipy.signal.windows``.

    Returns

    f : function
    The window function corresponding to the descriptor `w`

    See Also

    scipy.signal.windows

    Notes

    If no window function with a name matching the descriptor `w` is found in the
    module ``scipy.signal.windows``, the window function with the most similar name
    is be returned and a warning is issued.

    Example

    f = windowfcn('gaussian')
    f = windowfcn('gauss')

    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**zpdfr**(Signal=None, SamplingRate=None, UpperCutoff=None, LowerCutoff=None, Order=4.0)

```
    --DEPRECATED FUNCTION--

    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

# bspyclone.hr



## Modules

numpy

## Functions

**hr**(Signal=None, SamplingRate=1000.0)

```
ToDo.

Parameters
__________
ToDo : ToDo
        ToDo.

Returns
_______
kwrvals : dict
        A keyworded return values dict is returned with the following keys:
        IBI : ndarray
                ToDo.
        HR : ndarray
                ToDo.

See Also
________
        bvp.pulse
        ecg.ecg

Example
_______
ToDo

References
__________
.. ToDo
```

# bspyclone.peakd

index
d:\work\productioncode\clones\biosppy\bspyclone\peakd.py

```
.. module:: peadkd
   :platform: Unix, Windows
   :synopsis: This module provides various methods for peak detection.

.. moduleauthor:: Filipe Canento, Carlos Carreiras, Francisco David
```

## Modules

bspyclone.filt  numpy

## Functions

**sgndiff**(Signal=None, a=-1)

```
Determines Signal peaks.

Kwargs:
        Signal (array): input signal.

        a (int): Whether to return maxima (a = -1, the default) or minima (a = 1) points.

Kwrvals:
        Peak (array): peak indexes.

a is -1 as default, to detect peaks. if you want to detect minimum values, then change to 1

See Also:


Notes:


Example:


References:
        .. [1]
```

**ssf**(Signal=None, SamplingRate=1000.0, Filter={})

```
Determines Signal peaks.

Kwargs:
        Signal (array): input signal

        SamplingRate (float): Sampling frequency (Hz)

        Filter (dict): Filter coefficients

Kwrvals:
        Signal (array):

        Onset (array):

        SSF (array):


See Also:
```

```
    Notes:


    Example:


    References:
            .. [1] W.Zong, T.Heldt, G.B. Moody, and R.G. Mark,
                                "An Open-
    source Algorithm to Detect Onset of Arterial Blood Pressure Pulses",
                                Computers in Cardiology 2003; 30:259-262
```

# bspyclone.plux

index
d:\work\productioncode\clones\biosppy\bspyclone\plux.py

```
This module provides various functions to ...

Functions:

loadbpf()
```

## Modules

copy numpy pylab

## Classes

numpy.ndarray(__builtin__.object)

bparray

class **bparray**(numpy.ndarray)

```
#------------------------------------------------------------------------------------------------------------
--------------------
```

Method resolution order:
bparray
numpy.ndarray
__builtin__.object

---

Methods defined here:

**__array_finalize__**(self, obj)

**toADC**(self)

**toV**(self)

**tomV**(self)

**touS**(self)

---

Static methods defined here:

**__new__**(cls, ndarr, hdr={})

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

---

Methods inherited from numpy.ndarray:

**__abs__**(...)
x.__abs__() <==> abs(x)

**__add__**(...)
x.__add__(y) <==> x+y

**__and__**(...)

```
    x.__and__(y) <==> x&y

__array__(...)
    a.__array__(|dtype) -> reference if type unchanged, copy otherwise.

    Returns either a new reference to self if dtype is not given or a new array
    of provided data type if dtype is different from the current dtype of the
    array.

__array_prepare__(...)
    a.__array_prepare__(obj) -> Object of same type as ndarray object obj.

__array_wrap__(...)
    a.__array_wrap__(obj) -> Object of same type as ndarray object a.

__contains__(...)
    x.__contains__(y) <==> y in x

__copy__(...)
    a.__copy__([order])

    Return a copy of the array.

    Parameters
    ----------
    order : {'C', 'F', 'A'}, optional
        If order is 'C' (False) then the result is contiguous (default).
        If order is 'Fortran' (True) then the result has fortran order.
        If order is 'Any' (None) then the result has fortran order
        only if the array already is in fortran order.

__deepcopy__(...)
    a.__deepcopy__() -> Deep copy of array.

    Used if copy.deepcopy is called on an array.

__delitem__(...)
    x.__delitem__(y) <==> del x[y]

__delslice__(...)
    x.__delslice__(i, j) <==> del x[i:j]

    Use of negative indices is not supported.

__div__(...)
    x.__div__(y) <==> x/y

__divmod__(...)
    x.__divmod__(y) <==> divmod(x, y)

__eq__(...)
    x.__eq__(y) <==> x==y

__float__(...)
    x.__float__() <==> float(x)

__floordiv__(...)
    x.__floordiv__(y) <==> x//y

__ge__(...)
    x.__ge__(y) <==> x>=y

__getitem__(...)
    x.__getitem__(y) <==> x[y]

__getslice__(...)
    x.__getslice__(i, j) <==> x[i:j]

    Use of negative indices is not supported.
```

**__gt__**(...)
x.__gt__(y) <==> x>y

**__hex__**(...)
x.__hex__() <==> hex(x)

**__iadd__**(...)
x.__iadd__(y) <==> x+=y

**__iand__**(...)
x.__iand__(y) <==> x&=y

**__idiv__**(...)
x.__idiv__(y) <==> x/=y

**__ifloordiv__**(...)
x.__ifloordiv__(y) <==> x//y

**__ilshift__**(...)
x.__ilshift__(y) <==> x<<=y

**__imod__**(...)
x.__imod__(y) <==> x%=y

**__imul__**(...)
x.__imul__(y) <==> x*=y

**__index__**(...)
x[y:z] <==> x[y.__index__():z.__index__()]

**__int__**(...)
x.__int__() <==> int(x)

**__invert__**(...)
x.__invert__() <==> ~x

**__ior__**(...)
x.__ior__(y) <==> x|=y

**__ipow__**(...)
x.__ipow__(y) <==> x**=y

**__irshift__**(...)
x.__irshift__(y) <==> x>>=y

**__isub__**(...)
x.__isub__(y) <==> x-=y

**__iter__**(...)
x.__iter__() <==> iter(x)

**__itruediv__**(...)
x.__itruediv__(y) <==> x/y

**__ixor__**(...)
x.__ixor__(y) <==> x^=y

**__le__**(...)
x.__le__(y) <==> x<=y

**__len__**(...)
x.__len__() <==> len(x)

**__long__**(...)
x.__long__() <==> long(x)

**__lshift__**(...)
x.__lshift__(y) <==> x<<y

**__lt__**(...)
    x.__lt__(y) <==> x<y

**__mod__**(...)
    x.__mod__(y) <==> x%y

**__mul__**(...)
    x.__mul__(y) <==> x*y

**__ne__**(...)
    x.__ne__(y) <==> x!=y

**__neg__**(...)
    x.__neg__() <==> -x

**__nonzero__**(...)
    x.__nonzero__() <==> x != 0

**__oct__**(...)
    x.__oct__() <==> oct(x)

**__or__**(...)
    x.__or__(y) <==> x|y

**__pos__**(...)
    x.__pos__() <==> +x

**__pow__**(...)
    x.__pow__(y[, z]) <==> pow(x, y[, z])

**__radd__**(...)
    x.__radd__(y) <==> y+x

**__rand__**(...)
    x.__rand__(y) <==> y&x

**__rdiv__**(...)
    x.__rdiv__(y) <==> y/x

**__rdivmod__**(...)
    x.__rdivmod__(y) <==> divmod(y, x)

**__reduce__**(...)
    a.__reduce__()

    For pickling.

**__repr__**(...)
    x.__repr__() <==> repr(x)

**__rfloordiv__**(...)
    x.__rfloordiv__(y) <==> y//x

**__rlshift__**(...)
    x.__rlshift__(y) <==> y<<x

**__rmod__**(...)
    x.__rmod__(y) <==> y%x

**__rmul__**(...)
    x.__rmul__(y) <==> y*x

**__ror__**(...)
    x.__ror__(y) <==> y|x

**__rpow__**(...)
    y.__rpow__(x[, z]) <==> pow(x, y[, z])

**__rrshift__**(...)
x.__rrshift__(y) <==> y>>x

**__rshift__**(...)
x.__rshift__(y) <==> x>>y

**__rsub__**(...)
x.__rsub__(y) <==> y-x

**__rtruediv__**(...)
x.__rtruediv__(y) <==> y/x

**__rxor__**(...)
x.__rxor__(y) <==> y^x

**__setitem__**(...)
x.__setitem__(i, y) <==> x[i]=y

**__setslice__**(...)
x.__setslice__(i, j, y) <==> x[i:j]=y

Use  of negative indices is not supported.

**__setstate__**(...)
a.__setstate__(version, shape, dtype, isfortran, rawdata)

For unpickling.

Parameters
----------
version : int
    optional pickle version. If omitted defaults to 0.
shape : tuple
dtype : data-type
isFortran : bool
rawdata : string or list
    a binary string with the data (or a list if 'a' is an object array)

**__str__**(...)
x.__str__() <==> str(x)

**__sub__**(...)
x.__sub__(y) <==> x-y

**__truediv__**(...)
x.__truediv__(y) <==> x/y

**__xor__**(...)
x.__xor__(y) <==> x^y

**all**(...)
a.all(axis=None, out=None)

Returns True if all elements evaluate to True.

Refer to `numpy.all` for full documentation.

See Also
--------
numpy.all : equivalent function

**any**(...)
a.any(axis=None, out=None)

Returns True if any of the elements of `a` evaluate to True.

Refer to `numpy.any` for full documentation.

See Also
--------
numpy.any : equivalent function

**argmax**(...)

```
a.argmax(axis=None, out=None)

Return indices of the maximum values along the given axis.

Refer to `numpy.argmax` for full documentation.

See Also
--------
numpy.argmax : equivalent function
```

**argmin**(...)

```
a.argmin(axis=None, out=None)

Return indices of the minimum values along the given axis of `a`.

Refer to `numpy.argmin` for detailed documentation.

See Also
--------
numpy.argmin : equivalent function
```

**argpartition**(...)

```
a.argpartition(kth, axis=-1, kind='introselect', order=None)

Returns the indices that would partition this array.

Refer to `numpy.argpartition` for full documentation.

.. versionadded:: 1.8.0

See Also
--------
numpy.argpartition : equivalent function
```

**argsort**(...)

```
a.argsort(axis=-1, kind='quicksort', order=None)

Returns the indices that would sort this array.

Refer to `numpy.argsort` for full documentation.

See Also
--------
numpy.argsort : equivalent function
```

**astype**(...)

```
a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)

Copy of the array, cast to a specified type.

Parameters
----------
dtype : str or dtype
    Typecode or data-type to which the array is cast.
order : {'C', 'F', 'A', 'K'}, optional
    Controls the memory layout order of the result.
    'C' means C order, 'F' means Fortran order, 'A'
    means 'F' order if all the arrays are Fortran contiguous,
    'C' order otherwise, and 'K' means as close to the
    order the array elements appear in memory as possible.
    Default is 'K'.
casting : {'no', 'equiv', 'safe', 'same_kind', 'unsafe'}, optional
    Controls what kind of data casting may occur. Defaults to 'unsafe'
    for backwards compatibility.

      * 'no' means the data types should not be cast at all.
      * 'equiv' means only byte-order changes are allowed.
      * 'safe' means only casts which can preserve values are allowed.
      * 'same_kind' means only safe casts or casts within a kind,
        like float64 to float32, are allowed.
      * 'unsafe' means any data conversions may be done.
```

```
    subok : bool, optional
        If True, then sub-classes will be passed-through (default), otherwise
        the returned array will be forced to be a base-class array.
    copy : bool, optional
        By default, astype always returns a newly allocated array. If this
        is set to false, and the `dtype`, `order`, and `subok`
        requirements are satisfied, the input array is returned instead
        of a copy.

    Returns
    -------
    arr_t : ndarray
        Unless `copy` is False and the other conditions for returning the input
        array are satisfied (see description for `copy` input paramter), `arr_t`
        is a new array of the same shape as the input array, with dtype, order
        given by `dtype`, `order`.

    Notes
    -----
    Starting in NumPy 1.9, astype method now returns an error if the string
    dtype to cast to is not long enough in 'safe' casting mode to hold the max
    value of integer/float array that is being casted. Previously the casting
    was allowed even if the result was truncated.

    Raises
    ------
    ComplexWarning
        When casting from complex to float or int. To avoid this,
        one should use ``a.real.astype(t)``.

    Examples
    --------
    >>> x = np.array([1, 2, 2.5])
    >>> x
    array([ 1. ,  2. ,  2.5])

    >>> x.astype(int)
    array([1, 2, 2])
```

**byteswap**(...)

```
    a.byteswap(inplace)

    Swap the bytes of the array elements

    Toggle between low-endian and big-endian data representation by
    returning a byteswapped array, optionally swapped in-place.

    Parameters
    ----------
    inplace : bool, optional
        If ``True``, swap bytes in-place, default is ``False``.

    Returns
    -------
    out : ndarray
        The byteswapped array. If `inplace` is ``True``, this is
        a view to self.

    Examples
    --------
    >>> A = np.array([1, 256, 8755], dtype=np.int16)
    >>> map(hex, A)
    ['0x1', '0x100', '0x2233']
    >>> A.byteswap(True)
    array([  256,     1, 13090], dtype=int16)
    >>> map(hex, A)
    ['0x100', '0x1', '0x3322']

    Arrays of strings are not swapped

    >>> A = np.array(['ceg', 'fac'])
    >>> A.byteswap()
    array(['ceg', 'fac'],
          dtype='|S3')
```

**choose**(...)

```
a.choose(choices, out=None, mode='raise')

Use an index array to construct a new array from a set of choices.

Refer to `numpy.choose` for full documentation.

See Also
--------
numpy.choose : equivalent function
```

**clip**(...)

```
a.clip(a_min, a_max, out=None)

Return an array whose values are limited to ``[a_min, a_max]``.

Refer to `numpy.clip` for full documentation.

See Also
--------
numpy.clip : equivalent function
```

**compress**(...)

```
a.compress(condition, axis=None, out=None)

Return selected slices of this array along given axis.

Refer to `numpy.compress` for full documentation.

See Also
--------
numpy.compress : equivalent function
```

**conj**(...)

```
a.conj()

Complex-conjugate all elements.

Refer to `numpy.conjugate` for full documentation.

See Also
--------
numpy.conjugate : equivalent function
```

**conjugate**(...)

```
a.conjugate()

Return the complex conjugate, element-wise.

Refer to `numpy.conjugate` for full documentation.

See Also
--------
numpy.conjugate : equivalent function
```

**copy**(...)

```
a.copy(order='C')

Return a copy of the array.

Parameters
----------
order : {'C', 'F', 'A', 'K'}, optional
    Controls the memory layout of the copy. 'C' means C-order,
    'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,
    'C' otherwise. 'K' means match the layout of `a` as closely
    as possible. (Note that this function and :func:numpy.copy are very
    similar, but have different default values for their order=
    arguments.)

See also
--------
numpy.copy
```

```
numpy.copyto

Examples
--------
>>> x = np.array([[1,2,3],[4,5,6]], order='F')

>>> y = x.copy()

>>> x.fill(0)

>>> x
array([[0, 0, 0],
       [0, 0, 0]])

>>> y
array([[1, 2, 3],
       [4, 5, 6]])

>>> y.flags['C_CONTIGUOUS']
True
```

**cumprod**(...)

```
a.cumprod(axis=None, dtype=None, out=None)

Return the cumulative product of the elements along the given axis.

Refer to `numpy.cumprod` for full documentation.

See Also
--------
numpy.cumprod : equivalent function
```

**cumsum**(...)

```
a.cumsum(axis=None, dtype=None, out=None)

Return the cumulative sum of the elements along the given axis.

Refer to `numpy.cumsum` for full documentation.

See Also
--------
numpy.cumsum : equivalent function
```

**diagonal**(...)

```
a.diagonal(offset=0, axis1=0, axis2=1)

Return specified diagonals. In NumPy 1.9 the returned array is a
read-only view instead of a copy as in previous NumPy versions.  In
NumPy 1.10 the read-only restriction will be removed.

Refer to :func:`numpy.diagonal` for full documentation.

See Also
--------
numpy.diagonal : equivalent function
```

**dot**(...)

```
a.dot(b, out=None)

Dot product of two arrays.

Refer to `numpy.dot` for full documentation.

See Also
--------
numpy.dot : equivalent function

Examples
--------
>>> a = np.eye(2)
>>> b = np.ones((2, 2)) * 2
>>> a.dot(b)
array([[ 2.,  2.],
```

```
       [ 2.,  2.]])

This array method can be conveniently chained:

>>> a.dot(b).dot(b)
array([[ 8.,  8.],
       [ 8.,  8.]])
```

**dump**(...)

```
a.dump(file)

Dump a pickle of the array to the specified file.
The array can be read back with pickle.load or numpy.load.

Parameters
----------
file : str
    A string naming the dump file.
```

**dumps**(...)

```
a.dumps()

Returns the pickle of the array as a string.
pickle.loads or numpy.loads will convert the string back to an array.

Parameters
----------
None
```

**fill**(...)

```
a.fill(value)

Fill the array with a scalar value.

Parameters
----------
value : scalar
    All elements of `a` will be assigned this value.

Examples
--------
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([ 1.,  1.])
```

**flatten**(...)

```
a.flatten(order='C')

Return a copy of the array collapsed into one dimension.

Parameters
----------
order : {'C', 'F', 'A'}, optional
    Whether to flatten in C (row-major), Fortran (column-major) order,
    or preserve the C/Fortran ordering from `a`.
    The default is 'C'.

Returns
-------
y : ndarray
    A copy of the input array, flattened to one dimension.

See Also
--------
ravel : Return a flattened array.
flat : A 1-D flat iterator over the array.

Examples
```

```
--------
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()
array([1, 2, 3, 4])
>>> a.flatten('F')
array([1, 3, 2, 4])
```

**getfield**(...)

```
a.getfield(dtype, offset=0)

Returns a field of the given array as a certain type.

A field is a view of the array data with a given data-type. The values in
the view are determined by the given type and the offset into the current
array in bytes. The offset needs to be such that the view dtype fits in the
array dtype; for example an array of dtype complex128 has 16-byte elements.
If taking a view with a 32-bit integer (4 bytes), the offset needs to be
between 0 and 12 bytes.

Parameters
----------
dtype : str or dtype
    The data type of the view. The dtype size of the view can not be larger
    than that of the array itself.
offset : int
    Number of bytes to skip before beginning the element view.

Examples
--------
>>> x = np.diag([1.+1.j]*2)
>>> x[1, 1] = 2 + 4.j
>>> x
array([[ 1.+1.j,  0.+0.j],
       [ 0.+0.j,  2.+4.j]])
>>> x.getfield(np.float64)
array([[ 1.,  0.],
       [ 0.,  2.]])

By choosing an offset of 8 bytes we can select the complex part of the
array for our view:

>>> x.getfield(np.float64, offset=8)
array([[ 1.,  0.],
   [ 0.,  4.]])
```

**item**(...)

```
a.item(*args)

Copy an element of an array to a standard Python scalar and return it.

Parameters
----------
\*args : Arguments (variable number and type)

    * none: in this case, the method only works for arrays
      with one element (`a.size == 1`), which element is
      copied into a standard Python scalar object and returned.

    * int_type: this argument is interpreted as a flat index into
      the array, specifying which element to copy and return.

    * tuple of int_types: functions as does a single int_type argument,
      except that the argument is interpreted as an nd-index into the
      array.

Returns
-------
z : Standard Python scalar object
    A copy of the specified element of the array as a suitable
    Python scalar

Notes
-----
When the data type of `a` is longdouble or clongdouble, item() returns
```

```
    a scalar array object because there is no available Python scalar that
    would not lose information. Void arrays return a buffer object for item(),
    unless fields are defined, in which case a tuple is returned.

    `item` is very similar to a[args], except, instead of an array scalar,
    a standard Python scalar is returned. This can be useful for speeding up
    access to elements of the array and doing arithmetic on elements of the
    array using Python's optimized math.

    Examples
    --------
    >>> x = np.random.randint(9, size=(3, 3))
    >>> x
    array([[3, 1, 7],
           [2, 8, 3],
           [8, 5, 3]])
    >>> x.item(3)
    2
    >>> x.item(7)
    5
    >>> x.item((0, 1))
    1
    >>> x.item((2, 2))
    3
```

**itemset**(...)

```
    a.itemset(*args)

    Insert scalar into an array (scalar is cast to array's dtype, if possible)

    There must be at least 1 argument, and define the last argument
    as *item*.  Then, ``a.itemset(*args)`` is equivalent to but faster
    than ``a[args] = item``.  The item should be a scalar value and `args`
    must select a single item in the array `a`.

    Parameters
    ----------
    \*args : Arguments
        If one argument: a scalar, only used in case `a` is of size 1.
        If two arguments: the last argument is the value to be set
        and must be a scalar, the first argument specifies a single array
        element location. It is either an int or a tuple.

    Notes
    -----
    Compared to indexing syntax, `itemset` provides some speed increase
    for placing a scalar into a particular location in an `ndarray`,
    if you must do this.  However, generally this is discouraged:
    among other problems, it complicates the appearance of the code.
    Also, when using `itemset` (and `item`) inside a loop, be sure
    to assign the methods to a local variable to avoid the attribute
    look-up at each loop iteration.

    Examples
    --------
    >>> x = np.random.randint(9, size=(3, 3))
    >>> x
    array([[3, 1, 7],
           [2, 8, 3],
           [8, 5, 3]])
    >>> x.itemset(4, 0)
    >>> x.itemset((2, 2), 9)
    >>> x
    array([[3, 1, 7],
           [2, 0, 3],
           [8, 5, 9]])
```

**max**(...)

```
    a.max(axis=None, out=None)

    Return the maximum along a given axis.

    Refer to `numpy.amax` for full documentation.
```

```
    See Also
    --------
    numpy.amax : equivalent function
```

**mean**(...)

```
    a.mean(axis=None, dtype=None, out=None)

    Returns the average of the array elements along given axis.

    Refer to `numpy.mean` for full documentation.

    See Also
    --------
    numpy.mean : equivalent function
```

**min**(...)

```
    a.min(axis=None, out=None)

    Return the minimum along a given axis.

    Refer to `numpy.amin` for full documentation.

    See Also
    --------
    numpy.amin : equivalent function
```

**newbyteorder**(...)

```
    arr.newbyteorder(new_order='S')

    Return the array with the same data viewed with a different byte order.

    Equivalent to::

        arr.view(arr.dtype.newbytorder(new_order))

    Changes are also made in all fields and sub-arrays of the array data
    type.



    Parameters
    ----------
    new_order : string, optional
        Byte order to force; a value from the byte order specifications
        above. `new_order` codes can be any of::

         * 'S' - swap dtype from current to opposite endian
         * {'<', 'L'} - little endian
         * {'>', 'B'} - big endian
         * {'=', 'N'} - native order
         * {'|', 'I'} - ignore (no change to byte order)

        The default value ('S') results in swapping the current
        byte order. The code does a case-insensitive check on the first
        letter of `new_order` for the alternatives above.  For example,
        any of 'B' or 'b' or 'biggish' are valid to specify big-endian.


    Returns
    -------
    new_arr : array
        New array object with the dtype reflecting given change to the
        byte order.
```

**nonzero**(...)

```
    a.nonzero()

    Return the indices of the elements that are non-zero.

    Refer to `numpy.nonzero` for full documentation.

    See Also
    --------
```

```
    numpy.nonzero : equivalent function
```

**partition**(...)

```
    a.partition(kth, axis=-1, kind='introselect', order=None)

    Rearranges the elements in the array in such a way that value of the
    element in kth position is in the position it would be in a sorted array.
    All elements smaller than the kth element are moved before this element and
    all equal or greater are moved behind it. The ordering of the elements in
    the two partitions is undefined.

    .. versionadded:: 1.8.0

    Parameters
    ----------
    kth : int or sequence of ints
        Element index to partition by. The kth element value will be in its
        final sorted position and all smaller elements will be moved before it
        and all equal or greater elements behind it.
        The order all elements in the partitions is undefined.
        If provided with a sequence of kth it will partition all elements
        indexed by kth of them into their sorted position at once.
    axis : int, optional
        Axis along which to sort. Default is -1, which means sort along the
        last axis.
    kind : {'introselect'}, optional
        Selection algorithm. Default is 'introselect'.
    order : list, optional
        When `a` is an array with fields defined, this argument specifies
        which fields to compare first, second, etc.  Not all fields need be
        specified.

    See Also
    --------
    numpy.partition : Return a parititioned copy of an array.
    argpartition : Indirect partition.
    sort : Full sort.

    Notes
    -----
    See ``np.partition`` for notes on the different algorithms.

    Examples
    --------
    >>> a = np.array([3, 4, 2, 1])
    >>> a.partition(a, 3)
    >>> a
    array([2, 1, 3, 4])

    >>> a.partition((1, 3))
    array([1, 2, 3, 4])
```

**prod**(...)

```
    a.prod(axis=None, dtype=None, out=None)

    Return the product of the array elements over the given axis

    Refer to `numpy.prod` for full documentation.

    See Also
    --------
    numpy.prod : equivalent function
```

**ptp**(...)

```
    a.ptp(axis=None, out=None)

    Peak to peak (maximum - minimum) value along a given axis.

    Refer to `numpy.ptp` for full documentation.

    See Also
    --------
    numpy.ptp : equivalent function
```

**put**(...)

```
a.put(indices, values, mode='raise')

Set ``a.flat[n] = values[n]`` for all `n` in indices.

Refer to `numpy.put` for full documentation.

See Also
--------
numpy.put : equivalent function
```

**ravel**(...)

```
a.ravel([order])

Return a flattened array.

Refer to `numpy.ravel` for full documentation.

See Also
--------
numpy.ravel : equivalent function

ndarray.flat : a flat iterator on the array.
```

**repeat**(...)

```
a.repeat(repeats, axis=None)

Repeat elements of an array.

Refer to `numpy.repeat` for full documentation.

See Also
--------
numpy.repeat : equivalent function
```

**reshape**(...)

```
a.reshape(shape, order='C')

Returns an array containing the same data with a new shape.

Refer to `numpy.reshape` for full documentation.

See Also
--------
numpy.reshape : equivalent function
```

**resize**(...)

```
a.resize(new_shape, refcheck=True)

Change shape and size of array in-place.

Parameters
----------
new_shape : tuple of ints, or `n` ints
    Shape of resized array.
refcheck : bool, optional
    If False, reference count will not be checked. Default is True.

Returns
-------
None

Raises
------
ValueError
    If `a` does not own its own data or references or views to it exist,
    and the data memory must be changed.

SystemError
    If the `order` keyword argument is specified. This behaviour is a
    bug in NumPy.

See Also
```

```
--------
resize : Return a new array with the specified shape.

Notes
-----
This reallocates space for the data area if necessary.

Only contiguous arrays (data elements consecutive in memory) can be
resized.

The purpose of the reference count check is to make sure you
do not use this array as a buffer for another Python object and then
reallocate the memory. However, reference counts can increase in
other ways so if you are sure that you have not shared the memory
for this array with another Python object, then you may safely set
`refcheck` to False.

Examples
--------
Shrinking an array: array is flattened (in the order that the data are
stored in memory), resized, and reshaped:

>>> a = np.array([[0, 1], [2, 3]], order='C')
>>> a.resize((2, 1))
>>> a
array([[0],
       [1]])

>>> a = np.array([[0, 1], [2, 3]], order='F')
>>> a.resize((2, 1))
>>> a
array([[0],
       [2]])

Enlarging an array: as above, but missing entries are filled with zeros:

>>> b = np.array([[0, 1], [2, 3]])
>>> b.resize(2, 3) # new_shape parameter doesn't have to be a tuple
>>> b
array([[0, 1, 2],
       [3, 0, 0]])

Referencing an array prevents resizing...

>>> c = a
>>> a.resize((1, 1))
Traceback (most recent call last):
...
ValueError: cannot resize an array that has been referenced ...

Unless `refcheck` is False:

>>> a.resize((1, 1), refcheck=False)
>>> a
array([[0]])
>>> c
array([[0]])
```

**round**(...)

```
a.round(decimals=0, out=None)

Return `a` with each element rounded to the given number of decimals.

Refer to `numpy.around` for full documentation.

See Also
--------
numpy.around : equivalent function
```

**searchsorted**(...)

```
a.searchsorted(v, side='left', sorter=None)

Find indices where elements of v should be inserted in a to maintain order.
```

```
For full documentation, see `numpy.searchsorted`

See Also
--------
numpy.searchsorted : equivalent function
```

**setfield**(...)

```
a.setfield(val, dtype, offset=0)

Put a value into a specified place in a field defined by a data-type.

Place `val` into `a`'s field defined by `dtype` and beginning `offset`
bytes into the field.

Parameters
----------
val : object
    Value to be placed in field.
dtype : dtype object
    Data-type of the field in which to place `val`.
offset : int, optional
    The number of bytes into the field at which to place `val`.

Returns
-------
None

See Also
--------
getfield

Examples
--------
>>> x = np.eye(3)
>>> x.getfield(np.float64)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> x.setfield(3, np.int32)
>>> x.getfield(np.int32)
array([[3, 3, 3],
       [3, 3, 3],
       [3, 3, 3]])
>>> x
array([[  1.00000000e+000,   1.48219694e-323,   1.48219694e-323],
       [  1.48219694e-323,   1.00000000e+000,   1.48219694e-323],
       [  1.48219694e-323,   1.48219694e-323,   1.00000000e+000]])
>>> x.setfield(np.eye(3), np.int32)
>>> x
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

**setflags**(...)

```
a.setflags(write=None, align=None, uic=None)

Set array flags WRITEABLE, ALIGNED, and UPDATEIFCOPY, respectively.

These Boolean-valued flags affect how numpy interprets the memory
area used by `a` (see Notes below). The ALIGNED flag can only
be set to True if the data is actually aligned according to the type.
The UPDATEIFCOPY flag can never be set to True. The flag WRITEABLE
can only be set to True if the array owns its own memory, or the
ultimate owner of the memory exposes a writeable buffer interface,
or is a string. (The exception for string is made so that unpickling
can be done without copying memory.)

Parameters
----------
write : bool, optional
    Describes whether or not `a` can be written to.
align : bool, optional
    Describes whether or not `a` is aligned properly for its type.
uic : bool, optional
```

```
    Describes whether or not `a` is a copy of another "base" array.

Notes
-----
Array flags provide information about how the memory area used
for the array is to be interpreted. There are 6 Boolean flags
in use, only three of which can be changed by the user:
UPDATEIFCOPY, WRITEABLE, and ALIGNED.

WRITEABLE (W) the data area can be written to;

ALIGNED (A) the data and strides are aligned appropriately for the hardware
(as determined by the compiler);

UPDATEIFCOPY (U) this array is a copy of some other array (referenced
by .base). When this array is deallocated, the base array will be
updated with the contents of this array.

All flags can be accessed using their first (upper case) letter as well
as the full name.

Examples
--------
>>> y
array([[3, 1, 7],
       [2, 0, 0],
       [8, 5, 9]])
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
>>> y.setflags(write=0, align=0)
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : False
  ALIGNED : False
  UPDATEIFCOPY : False
>>> y.setflags(uic=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot set UPDATEIFCOPY flag to True
```

**sort**(...)

```
a.sort(axis=-1, kind='quicksort', order=None)

Sort an array, in-place.

Parameters
----------
axis : int, optional
    Axis along which to sort. Default is -1, which means sort along the
    last axis.
kind : {'quicksort', 'mergesort', 'heapsort'}, optional
    Sorting algorithm. Default is 'quicksort'.
order : list, optional
    When `a` is an array with fields defined, this argument specifies
    which fields to compare first, second, etc.  Not all fields need be
    specified.

See Also
--------
numpy.sort : Return a sorted copy of an array.
argsort : Indirect sort.
lexsort : Indirect stable sort on multiple keys.
searchsorted : Find elements in sorted array.
partition: Partial sort.

Notes
-----
```

```
    See ``sort`` for notes on the different sorting algorithms.

    Examples
    --------
    >>> a = np.array([[1,4], [3,1]])
    >>> a.sort(axis=1)
    >>> a
    array([[1, 4],
           [1, 3]])
    >>> a.sort(axis=0)
    >>> a
    array([[1, 3],
           [1, 4]])

    Use the `order` keyword to specify a field to use when sorting a
    structured array:

    >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])
    >>> a.sort(order='y')
    >>> a
    array([('c', 1), ('a', 2)],
          dtype=[('x', '|S1'), ('y', '<i4')])
```

**squeeze**(...)

```
    a.squeeze(axis=None)

    Remove single-dimensional entries from the shape of `a`.

    Refer to `numpy.squeeze` for full documentation.

    See Also
    --------
    numpy.squeeze : equivalent function
```

**std**(...)

```
    a.std(axis=None, dtype=None, out=None, ddof=0)

    Returns the standard deviation of the array elements along given axis.

    Refer to `numpy.std` for full documentation.

    See Also
    --------
    numpy.std : equivalent function
```

**sum**(...)

```
    a.sum(axis=None, dtype=None, out=None)

    Return the sum of the array elements over the given axis.

    Refer to `numpy.sum` for full documentation.

    See Also
    --------
    numpy.sum : equivalent function
```

**swapaxes**(...)

```
    a.swapaxes(axis1, axis2)

    Return a view of the array with `axis1` and `axis2` interchanged.

    Refer to `numpy.swapaxes` for full documentation.

    See Also
    --------
    numpy.swapaxes : equivalent function
```

**take**(...)

```
    a.take(indices, axis=None, out=None, mode='raise')

    Return an array formed from the elements of `a` at the given indices.

    Refer to `numpy.take` for full documentation.
```

```
    See Also
    --------
    numpy.take : equivalent function
```

**tobytes**(...)

```
    a.tobytes(order='C')

    Construct Python bytes containing the raw data bytes in the array.

    Constructs Python bytes showing a copy of the raw contents of
    data memory. The bytes object can be produced in either 'C' or 'Fortran',
    or 'Any' order (the default is 'C'-order). 'Any' order means C-order
    unless the F_CONTIGUOUS flag in the array is set, in which case it
    means 'Fortran' order.

    .. versionadded:: 1.9.0

    Parameters
    ----------
    order : {'C', 'F', None}, optional
        Order of the data for multidimensional arrays:
        C, Fortran, or the same as for the original array.

    Returns
    -------
    s : bytes
        Python bytes exhibiting a copy of `a`'s raw data.

    Examples
    --------
    >>> x = np.array([[0, 1], [2, 3]])
    >>> x.tobytes()
    b'\x00\x00\x00\x00\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00'
    >>> x.tobytes('C') == x.tobytes()
    True
    >>> x.tobytes('F')
    b'\x00\x00\x00\x00\x02\x00\x00\x00\x01\x00\x00\x00\x03\x00\x00\x00'
```

**tofile**(...)

```
    a.tofile(fid, sep="", format="%s")

    Write array to a file as text or binary (default).

    Data is always written in 'C' order, independent of the order of `a`.
    The data produced by this method can be recovered using the function
    fromfile().

    Parameters
    ----------
    fid : file or str
        An open file object, or a string containing a filename.
    sep : str
        Separator between array items for text output.
        If "" (empty), a binary file is written, equivalent to
        ``file.write(a.tobytes())``.
    format : str
        Format string for text file output.
        Each entry in the array is formatted to text by first converting
        it to the closest Python type, and then using "format" % item.

    Notes
    -----
    This is a convenience function for quick storage of array data.
    Information on endianness and precision is lost, so this method is not a
    good choice for files intended to archive data or transport data between
    machines with different endianness. Some of these problems can be overcome
    by outputting the data as text files, at the expense of speed and file
    size.
```

**tolist**(...)

```
    a.tolist()
```

```
Return the array as a (possibly nested) list.

Return a copy of the array data as a (nested) Python list.
Data items are converted to the nearest compatible Python type.

Parameters
----------
none

Returns
-------
y : list
    The possibly nested list of array elements.

Notes
-----
The array may be recreated, ``a = np.array(a.tolist())``.

Examples
--------
>>> a = np.array([1, 2])
>>> a.tolist()
[1, 2]
>>> a = np.array([[1, 2], [3, 4]])
>>> list(a)
[array([1, 2]), array([3, 4])]
>>> a.tolist()
[[1, 2], [3, 4]]
```

**tostring**(...)

```
a.tostring(order='C')

Construct Python bytes containing the raw data bytes in the array.

Constructs Python bytes showing a copy of the raw contents of
data memory. The bytes object can be produced in either 'C' or 'Fortran',
or 'Any' order (the default is 'C'-order). 'Any' order means C-order
unless the F_CONTIGUOUS flag in the array is set, in which case it
means 'Fortran' order.

This function is a compatibility alias for tobytes. Despite its name it returns bytes not strings.

Parameters
----------
order : {'C', 'F', None}, optional
    Order of the data for multidimensional arrays:
    C, Fortran, or the same as for the original array.

Returns
-------
s : bytes
    Python bytes exhibiting a copy of `a`'s raw data.

Examples
--------
>>> x = np.array([[0, 1], [2, 3]])
>>> x.tobytes()
b'\x00\x00\x00\x00\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00'
>>> x.tobytes('C') == x.tobytes()
True
>>> x.tobytes('F')
b'\x00\x00\x00\x00\x02\x00\x00\x00\x01\x00\x00\x00\x03\x00\x00\x00'
```

**trace**(...)

```
a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

Return the sum along diagonals of the array.

Refer to `numpy.trace` for full documentation.

See Also
--------
numpy.trace : equivalent function
```

**transpose**(...)

```
a.transpose(*axes)

Returns a view of the array with axes transposed.

For a 1-D array, this has no effect. (To change between column and
row vectors, first cast the 1-D array into a matrix object.)
For a 2-D array, this is the usual matrix transpose.
For an n-D array, if axes are given, their order indicates how the
axes are permuted (see Examples). If axes are not provided and
``a.shape = (i[0], i[1], ... i[n-2], i[n-1])``, then
``a.transpose().shape = (i[n-1], i[n-2], ... i[1], i[0])``.

Parameters
----------
axes : None, tuple of ints, or `n` ints

 * None or no argument: reverses the order of the axes.

 * tuple of ints: `i` in the `j`-th place in the tuple means `a`'s
   `i`-th axis becomes `a.transpose()`'s `j`-th axis.

 * `n` ints: same as an n-tuple of the same ints (this form is
   intended simply as a "convenience" alternative to the tuple form)

Returns
-------
out : ndarray
    View of `a`, with axes suitably permuted.

See Also
--------
ndarray.T : Array property returning the array transposed.

Examples
--------
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.transpose()
array([[1, 3],
       [2, 4]])
>>> a.transpose((1, 0))
array([[1, 3],
       [2, 4]])
>>> a.transpose(1, 0)
array([[1, 3],
       [2, 4]])
```

**var**(...)

```
a.var(axis=None, dtype=None, out=None, ddof=0)

Returns the variance of the array elements, along given axis.

Refer to `numpy.var` for full documentation.

See Also
--------
numpy.var : equivalent function
```

**view**(...)

```
a.view(dtype=None, type=None)

New view of array with the same data.

Parameters
----------
dtype : data-type or ndarray sub-class, optional
    Data-type descriptor of the returned view, e.g., float32 or int16. The
    default, None, results in the view having the same data-type as `a`.
    This argument can also be specified as an ndarray sub-class, which
    then specifies the type of the returned object (this is equivalent to
```

```
    setting the ``type`` parameter).
type : Python type, optional
    Type of the returned view, e.g., ndarray or matrix.  Again, the
    default None results in type preservation.

Notes
-----
``a.view()`` is used two different ways:

``a.view(some_dtype)`` or ``a.view(dtype=some_dtype)`` constructs a view
of the array's memory with a different data-type.  This can cause a
reinterpretation of the bytes of memory.

``a.view(ndarray_subclass)`` or ``a.view(type=ndarray_subclass)`` just
returns an instance of `ndarray_subclass` that looks at the same array
(same shape, dtype, etc.)  This does not cause a reinterpretation of the
memory.

For ``a.view(some_dtype)``, if ``some_dtype`` has a different number of
bytes per entry than the previous dtype (for example, converting a
regular array to a structured array), then the behavior of the view
cannot be predicted just from the superficial appearance of ``a`` (shown
by ``print(a)``). It also depends on exactly how ``a`` is stored in
memory. Therefore if ``a`` is C-ordered versus fortran-ordered, versus
defined as a slice or transpose, etc., the view may give different
results.


Examples
--------
>>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])

Viewing array data using a different type and dtype:

>>> y = x.view(dtype=np.int16, type=np.matrix)
>>> y
matrix([[513]], dtype=int16)
>>> print type(y)
<class 'numpy.matrixlib.defmatrix.matrix'>

Creating a view on a structured array so it can be used in calculations

>>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])
>>> xv = x.view(dtype=np.int8).reshape(-1,2)
>>> xv
array([[1, 2],
       [3, 4]], dtype=int8)
>>> xv.mean(0)
array([ 2.,  3.])

Making changes to the view changes the underlying array

>>> xv[0,1] = 20
>>> print x
[(1, 20) (3, 4)]

Using a view to convert an array to a record array:

>>> z = x.view(np.recarray)
>>> z.a
array([1], dtype=int8)

Views share data:

>>> x[0] = (9, 10)
>>> z[0]
(9, 10)

Views that change the dtype size (bytes per entry) should normally be
avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

>>> x = np.array([[1,2,3],[4,5,6]], dtype=np.int16)
>>> y = x[:, 0:2]
>>> y
```

```
array([[1, 2],
       [4, 5]], dtype=int16)
>>> y.view(dtype=[('width', np.int16), ('length', np.int16)])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: new type not compatible with array.
>>> z = y.copy()
>>> z.view(dtype=[('width', np.int16), ('length', np.int16)])
array([[(1, 2)],
       [(4, 5)]], dtype=[('width', '<i2'), ('length', '<i2')])
```

---

Data descriptors inherited from numpy.ndarray:

**T**

```
Same as transpose(), except that self is returned if
self.ndim < 2.

Examples
--------
>>> x = np.array([[1.,2.],[3.,4.]])
>>> x
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> x.T
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> x = np.array([1.,2.,3.,4.])
>>> x
array([ 1.,  2.,  3.,  4.])
>>> x.T
array([ 1.,  2.,  3.,  4.])
```

**__array_interface__**

```
Array protocol: Python side.
```

**__array_priority__**

```
Array priority.
```

**__array_struct__**

```
Array protocol: C-struct side.
```

**base**

```
Base object if memory is from some other object.

Examples
--------
The base of an array that owns its memory is None:

>>> x = np.array([1,2,3,4])
>>> x.base is None
True

Slicing creates a view, whose memory is shared with x:

>>> y = x[2:]
>>> y.base is x
True
```

**ctypes**

```
An object to simplify the interaction of the array with the ctypes
module.

This attribute creates an object that makes it easier to use arrays
when calling shared libraries with the ctypes module. The returned
object has, among others, data, shape, and strides attributes (see
Notes below) which themselves return ctypes objects that can be used
as arguments to a shared library.

Parameters
----------
None
```

```
Returns
-------
c : Python object
    Possessing attributes data, shape, strides, etc.

See Also
--------
numpy.ctypeslib

Notes
-----
Below are the public attributes of this object which were documented
in "Guide to NumPy" (we have omitted undocumented public attributes,
as well as documented private attributes):

* data: A pointer to the memory area of the array as a Python integer.
  This memory area may contain data that is not aligned, or not in correct
  byte-order. The memory area may not even be writeable. The array
  flags and data-type of this array should be respected when passing this
  attribute to arbitrary C-code to avoid trouble that can include Python
  crashing. User Beware! The value of this attribute is exactly the same
  as self._array_interface_['data'][0].

* shape (c_intp*self.ndim): A ctypes array of length self.ndim where
  the basetype is the C-integer corresponding to dtype('p') on this
  platform. This base-type could be c_int, c_long, or c_longlong
  depending on the platform. The c_intp type is defined accordingly in
  numpy.ctypeslib. The ctypes array contains the shape of the underlying
  array.

* strides (c_intp*self.ndim): A ctypes array of length self.ndim where
  the basetype is the same as for the shape attribute. This ctypes array
  contains the strides information from the underlying array. This strides
  information is important for showing how many bytes must be jumped to
  get to the next element in the array.

* data_as(obj): Return the data pointer cast to a particular c-types object.
  For example, calling self._as_parameter_ is equivalent to
  data_as(ctypes.c_void_p). Perhaps you want to use the data as a
  pointer to a ctypes array of floating-point data:
  data_as(ctypes.POINTER(ctypes.c_double)).

* shape_as(obj): Return the shape tuple as an array of some other c-types
  type. For example: shape_as(ctypes.c_short).

* strides_as(obj): Return the strides tuple as an array of some other
  c-types type. For example: strides_as(ctypes.c_longlong).

Be careful using the ctypes attribute - especially on temporary
arrays or arrays constructed on the fly. For example, calling
``(a+b).ctypes.data_as(ctypes.c_void_p)`` returns a pointer to memory
that is invalid because the array created as (a+b) is deallocated
before the next Python statement. You can avoid this problem using
either ``c=a+b`` or ``ct=(a+b).ctypes``. In the latter case, ct will
hold a reference to the array until ct is deleted or re-assigned.

If the ctypes module is not available, then the ctypes attribute
of array objects still returns something useful, but ctypes objects
are not returned and errors may be raised instead. In particular,
the object will still have the as parameter attribute which will
return an integer equal to the data attribute.

Examples
--------
>>> import ctypes
>>> x
array([[0, 1],
       [2, 3]])
>>> x.ctypes.data
30439712
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_long))
<ctypes.LP_c_long object at 0x01F01300>
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_long)).contents
c_long(0)
```

```
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_longlong)).contents
c_longlong(4294967296L)
>>> x.ctypes.shape
<numpy.core._internal.c_long_Array_2 object at 0x01FFD580>
>>> x.ctypes.shape_as(ctypes.c_long)
<numpy.core._internal.c_long_Array_2 object at 0x01FCE620>
>>> x.ctypes.strides
<numpy.core._internal.c_long_Array_2 object at 0x01FCE620>
>>> x.ctypes.strides_as(ctypes.c_longlong)
<numpy.core._internal.c_longlong_Array_2 object at 0x01F01300>
```

**data**

```
Python buffer object pointing to the start of the array's data.
```

**dtype**

```
Data-type of the array's elements.

Parameters
----------
None

Returns
-------
d : numpy dtype object

See Also
--------
numpy.dtype

Examples
--------
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int32')
>>> type(x.dtype)
<type 'numpy.dtype'>
```

**flags**

```
Information about the memory layout of the array.

Attributes
----------
C_CONTIGUOUS (C)
    The data is in a single, C-style contiguous segment.
F_CONTIGUOUS (F)
    The data is in a single, Fortran-style contiguous segment.
OWNDATA (O)
    The array owns the memory it uses or borrows it from another object.
WRITEABLE (W)
    The data area can be written to.  Setting this to False locks
    the data, making it read-only.  A view (slice, etc.) inherits WRITEABLE
    from its base array at creation time, but a view of a writeable
    array may be subsequently locked while the base array remains writeable.
    (The opposite is not true, in that a view of a locked array may not
    be made writeable.  However, currently, locking a base object does not
    lock any views that already reference it, so under that circumstance it
    is possible to alter the contents of a locked array via a previously
    created writeable view onto it.)  Attempting to change a non-writeable
    array raises a RuntimeError exception.
ALIGNED (A)
    The data and all elements are aligned appropriately for the hardware.
UPDATEIFCOPY (U)
    This array is a copy of some other array. When this array is
    deallocated, the base array will be updated with the contents of
    this array.
FNC
    F_CONTIGUOUS and not C_CONTIGUOUS.
FORC
    F_CONTIGUOUS or C_CONTIGUOUS (one-segment test).
BEHAVED (B)
    ALIGNED and WRITEABLE.
```

```
    CARRAY (CA)
        BEHAVED and C_CONTIGUOUS.
    FARRAY (FA)
        BEHAVED and F_CONTIGUOUS and not C_CONTIGUOUS.

    Notes
    -----
    The `flags` object can be accessed dictionary-like (as in ``a.flags['WRITEABLE']``),
    or by using lowercased attribute names (as in ``a.flags.writeable``). Short flag
    names are only supported in dictionary access.

    Only the UPDATEIFCOPY, WRITEABLE, and ALIGNED flags can be changed by
    the user, via direct assignment to the attribute or dictionary entry,
    or by calling `ndarray.setflags`.

    The array flags cannot be set arbitrarily:

    - UPDATEIFCOPY can only be set ``False``.
    - ALIGNED can only be set ``True`` if the data is truly aligned.
    - WRITEABLE can only be set ``True`` if the array owns its own memory
      or the ultimate owner of the memory exposes a writeable buffer
      interface or is a string.

    Arrays can be both C-style and Fortran-style contiguous simultaneously.
    This is clear for 1-dimensional arrays, but can also be true for higher
    dimensional arrays.

    Even for contiguous arrays a stride for a given dimension
    ``arr.strides[dim]`` may be *arbitrary* if ``arr.shape[dim] == 1``
    or the array has no elements.
    It does *not* generally hold that ``self.strides[-1] == self.itemsize``
    for C-style contiguous arrays or ``self.strides[0] == self.itemsize`` for
    Fortran-style contiguous arrays is true.
```

**flat**

```
    A 1-D iterator over the array.

    This is a `numpy.flatiter` instance, which acts similarly to, but is not
    a subclass of, Python's built-in iterator object.

    See Also
    --------
    flatten : Return a copy of the array collapsed into one dimension.

    flatiter

    Examples
    --------
    >>> x = np.arange(1, 7).reshape(2, 3)
    >>> x
    array([[1, 2, 3],
           [4, 5, 6]])
    >>> x.flat[3]
    4
    >>> x.T
    array([[1, 4],
           [2, 5],
           [3, 6]])
    >>> x.T.flat[3]
    5
    >>> type(x.flat)
    <type 'numpy.flatiter'>

    An assignment example:

    >>> x.flat = 3; x
    array([[3, 3, 3],
           [3, 3, 3]])
    >>> x.flat[[1,4]] = 1; x
    array([[3, 1, 3],
           [3, 1, 3]])
```

**imag**

```
    The imaginary part of the array.
```

```
Examples
--------
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.imag
array([ 0.        ,  0.70710678])
>>> x.imag.dtype
dtype('float64')
```

**itemsize**

```
Length of one array element in bytes.

Examples
--------
>>> x = np.array([1,2,3], dtype=np.float64)
>>> x.itemsize
8
>>> x = np.array([1,2,3], dtype=np.complex128)
>>> x.itemsize
16
```

**nbytes**

```
Total bytes consumed by the elements of the array.

Notes
-----
Does not include memory consumed by non-element attributes of the
array object.

Examples
--------
>>> x = np.zeros((3,5,2), dtype=np.complex128)
>>> x.nbytes
480
>>> np.prod(x.shape) * x.itemsize
480
```

**ndim**

```
Number of array dimensions.

Examples
--------
>>> x = np.array([1, 2, 3])
>>> x.ndim
1
>>> y = np.zeros((2, 3, 4))
>>> y.ndim
3
```

**real**

```
The real part of the array.

Examples
--------
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.real
array([ 1.        ,  0.70710678])
>>> x.real.dtype
dtype('float64')

See Also
--------
numpy.real : equivalent function
```

**shape**

```
Tuple of array dimensions.

Notes
-----
May be used to "reshape" the array, as long as this would not
require a change in the total number of elements

Examples
```

```
--------
>>> x = np.array([1, 2, 3, 4])
>>> x.shape
(4,)
>>> y = np.zeros((2, 3, 4))
>>> y.shape
(2, 3, 4)
>>> y.shape = (3, 8)
>>> y
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> y.shape = (3, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: total size of new array must be unchanged
```

**size**

```
Number of elements in the array.

Equivalent to ``np.prod(a.shape)``, i.e., the product of the array's
dimensions.

Examples
--------
>>> x = np.zeros((3, 5, 2), dtype=np.complex128)
>>> x.size
30
>>> np.prod(x.shape)
30
```

**strides**

```
Tuple of bytes to step in each dimension when traversing an array.

The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `a`
is::

    offset = sum(np.array(i) * a.strides)

A more detailed explanation of strides can be found in the
"ndarray.rst" file in the NumPy reference guide.

Notes
-----
Imagine an array of 32-bit integers (each 4 bytes)::

  x = np.array([[0, 1, 2, 3, 4],
                [5, 6, 7, 8, 9]], dtype=np.int32)

This array is stored in memory as 40 bytes, one after the other
(known as a contiguous block of memory).  The strides of an array tell
us how many bytes we have to skip in memory to move to the next position
along a certain axis.  For example, we have to skip 4 bytes (1 value) to
move to the next column, but 20 bytes (5 values) to get to the same
position in the next row.  As such, the strides for the array `x` will be
``(20, 4)``.

See Also
--------
numpy.lib.stride_tricks.as_strided

Examples
--------
>>> y = np.reshape(np.arange(2*3*4), (2,3,4))
>>> y
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>> y.strides
(48, 16, 4)
>>> y[1,1,1]
```

```
17
>>> offset=sum(y.strides * np.array((1,1,1)))
>>> offset/y.itemsize
17

>>> x = np.reshape(np.arange(5*6*7*8), (5,6,7,8)).transpose(2,3,1,0)
>>> x.strides
(32, 4, 224, 1344)
>>> i = np.array([3,5,2,2])
>>> offset = sum(i * x.strides)
>>> x[3,5,2,2]
813
>>> offset / x.itemsize
813
```

## Functions

**loadbpf**(fname)

```
Loads biosignal data.

Parameters
__________
fname : string
        input filename

Returns
_______
bpd : bparray
        output biosignal array

See Also
________
        np.loadtxt

Notes
_____


Example
_______


References
__________
.. [1]

http://
```

# bspyclone.resp.resp

index
d:\work\productioncode\clones\biosppy\bspyclone\resp\resp.py

```
.. module:: resp
   :platform: Unix, Windows
   :synopsis: This module provides various functions to handle RESP signals.

.. moduleauthor:: Filipe Canento
```

## Modules

numpy scipy unittest
pylab sys

## Functions

**features**(Signal=None, SamplingRate=1000.0, Filter={})

```
    Retrieves relevant RESP signal features.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): filter parameters.

    Kwrvals:
        Signal (array): output filtered signal.

        FR (array): instant respiratory frequency (Hz)

        ZC (array): zero crossings indexes

        mean (float): mean

        std (float): standard deviation

        var (float): variance

        skew (ndarry): skewness

        kurtosis (array): kurtosis

        ad (float): absolute deviation

    Configurable fields:
    {"name": "resp.features", "config": {"SamplingRate": "1000."}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "FR", "ZC", "mean", "sto

    See Also:
        filt

        tls.statsf

    Notes:


    Example:


    References:
        .. [1]
```

**filt**(Signal=None, SamplingRate=1000.0, UpperCutoff=0.35, LowerCutoff=0.1, Order=2.0)

```
    Filters an input RESP signal.

    If only input signal is provide, it returns the filtered RESP signal
    assuming a 1000Hz sampling frequency and ...

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        UpperCutoff (float): Low-pass filter cutoff frequency (Hz).

        LowerCutoff (float): High-pass filter cutoff frequency (Hz).

        Order (int): Filter order.

    Kwrvals:
        Signal (array): output filtered signal.

    Configurable fields:
    {"name": "resp.filt", "config": {"UpperCutoff": "0.35", "SamplingRate": "1000.", "LowerCutoff": "0.1", "Order": "2."}, "inputs": ["Signal"],

    See Also:
            flt.zpdfr

    Notes:
```

```
    Example:
        Signal = load(...)
        SamplingRate = ...
        res = filt(Signal=Signal, SamplingRate=SamplingRate)
        plot(res['Signal'])

    References:
        .. [1]
```

**resp**(Signal=None, SamplingRate=1000.0, Filter=())

```
    Respiratory signal information.

    Kwargs:
        Signal (array): input signal.

        SamplingRate (float): Sampling frequency (Hz).

        Filter (dict): filter parameters.

    Kwrvals:
        Signal (array): output filtered signal.

        FR (array): instant respiratory frequency (Hz)

        ZC (array): zero crossings indexes

    Configurable fields:{"name": "resp.resp", "config": {"SamplingRate": "1000.0"}, "inputs": ["Signal", "Filter"], "outputs": ["Signal", "FR", "

    See Also:
        filt

    Notes:


    Example:


    References:
        .. [1]
```

# bspyclone.sync

index
d:\work\productioncode\clones\biosppy\bspyclone\sync.py

This module provides various functions to ...

Functions:

step()
threshold()
match()
pair()

## Modules

pylab

## Functions

**match**(Signal=None, Window=None)

.

Parameters
_____
Signal : ndarray
Input signal data.
Window :

Returns
_____
kwrvals : dict
A keyworded return values dict is returned with the following keys:
Event :

See Also
_____
pl.convolve

Notes
_____

Example
_____

**pair**(x, y)

.

Parameters
_____
x : array

y : array

Returns
_____
x : array

y : array

```
See Also
________


Notes
_____


Example
_______
```

**step**(Signal=None, Shift='>')

```
Find the indexes where a dirac, unit step or unit pulse function in the input
signal rises or falls.

This implementation is based on the discrete difference of the input signal.

Parameters
__________
Signal : ndarray
        Input signal data with a dirac, unit step or unit pulse function.
Shift : str
        Direction of detection, `>` for rise or `<` for fall.
        Default: `>`

Returns
_______
kwrvals : dict
        A keyworded return values dict is returned with the following keys:
        Event : ndarray
                The indexes within the input signal `Signal` where the `Shift` was detected.

See Also
________
        sync.threshold
        sync.match

Example
_______
x = zeros(6)
x[2:4] = 1
plot(x)
vlines(step(x)['Events'],min(x),max(x),'r','dashed')
vlines(step(x,'<')['Events'],min(x),max(x),'g','dashed')
legend(('unit pulse','rise', 'fall'))

References
__________
.. [1] Wikipedia, "Dirac Delta Function".
http://en.wikipedia.org/wiki/Dirac_delta_function
.. [2] Wikipedia, "Heaviside Step Function".
http://en.wikipedia.org/wiki/Heaviside_step_function
.. [3] Wikipedia, "Unit Pulse Function".
http://en.wikipedia.org/wiki/Rectangular_function
```

**threshold**(Signal=None, Threshold=0.1, Shift='>')

```
Find the indexes where a the input signal rises above or falls bellow a given
threshold.

Parameters
__________
Signal : ndarray
        Input signal data.
Threshold : int, float
        Detection threshold. If this is an `int` greater or equal than one it will be
        directly used as the threshold. If this is a `float`, the threshold is
        computed as the `Threshold` percentage of the signal span.
```

```
        Default: .1
Shift : str
        Direction of detection, `>` for rise or `<` for fall.
        Default: `>`

Returns
_______
kwrvals : dict
        A keyworded return values dict is returned with the following keys:
        Event : ndarray
                The indexes within the input signal `Signal` where a `Shift` with respect to
                `Threshold` was detected.

See Also
________
        sync.step
        sync.match

Notes
_____
This method is primarily designed to detect shifts in the input signal with
respect to a threshold computed as a percentage of the signal span.

Example
_______
x = zeros(6)
x[2:4] = 1
plot(x)
th=.5
plot(ones(len(x))*th,'k--')
vlines(threshold(x,th)['Event'],min(x),max(x),'r','dashed')
vlines(threshold(x,th,'<')['Event'],min(x),max(x),'g','dashed')
legend(('unit pulse','rise', 'fall'))
```

# bspyclone.tools

index
d:\work\productioncode\clones\biosppy\bspyclone\tools.py

```
This module provides various auxiliary functions.

Functions:
```



## Modules

locale os sys
numpy pylab

## Functions

**biplot**(m)

**fileparts**(file)

```
    #--------------------------------------------------------------------------
    ------------------------------------------
```

**format_num**(num)

```
    Format a number according to given places.
    Adds commas, etc. Will truncate floats into ints!
```

**fullfile**(*args)

```
    #--------------------------------------------------------------------------
    ------------------------------------------
```

**get_max_width**(table, index)

```
    Get the maximum width of the given column index
```

**pprint_table**(out, table)

```
    Prints out a table of data, padded for alignment
    @param out: Output stream (file-like object)
    @param table: The table to print. A list of lists.
    Each row must have the same number of columns.
```

**scale**(x)

```
    #--------------------------------------------------------------------------
    ------------------------------------------
```

**statsf**(Signal=None)

```
    Determine various statistical features of a given input signal.

    Parameters
    __________
    Signal : array
            input signal

    Returns
    _______
    kwrvals : dict
            A keyworded return values dict is returned with the following keys:
                    'mean' : float
                            mean
                    'std' : float
                            standard deviation
                    'var' : float
                            variance
                    'skew' : ndarry
                            skewness
                    'kurtosis' : array
                            kurtosis
                    'ad' : float
                            absolute deviation

    See Also
    ________
            np.mean
            np.std
            np.var
            np.skew
            scipy.stats.skew
            scipy.stats.kurtosis

    Notes
    _____


    Example
    _______


    References
    __________
    .. [1]
    .. [2]
    http://
```

**walktree**(top, callback=<function disp>)

```
    recursively descend the directory tree rooted at top,
    calling the callback function for each regular file

    EXTRACTED FROM:
    Python Library
    http://docs.python.org/library/stat.html
```

**zerocross**(Signal=None)

```
    Determine the indexes where the signal crosses zero.

    Parameters
    __________
```

```
    signal : array
            input signal

    Returns
    _______
    kwrvals : dict
            A keyworded return values dict is returned with the following keys:
                    'zc' : array
                            indexes of the zero crossings.

    See Also
    ________


    Notes
    _____


    Example
    _______


    References
    __________
    .. [1]
    .. [2]
    http://
```

## Data

**SF_APPEND** = 262144
**SF_ARCHIVED** = 65536
**SF_IMMUTABLE** = 131072
**SF_NOUNLINK** = 1048576
**SF_SNAPSHOT** = 2097152
**ST_ATIME** = 7
**ST_CTIME** = 9
**ST_DEV** = 2
**ST_GID** = 5
**ST_INO** = 1
**ST_MODE** = 0
**ST_MTIME** = 8
**ST_NLINK** = 3
**ST_SIZE** = 6
**ST_UID** = 4
**S_ENFMT** = 1024
**S_IEXEC** = 64
**S_IFBLK** = 24576
**S_IFCHR** = 8192
**S_IFDIR** = 16384
**S_IFIFO** = 4096
**S_IFLNK** = 40960
**S_IFREG** = 32768
**S_IFSOCK** = 49152

**S_IREAD** = 256
**S_IRGRP** = 32
**S_IROTH** = 4
**S_IRUSR** = 256
**S_IRWXG** = 56
**S_IRWXO** = 7
**S_IRWXU** = 448
**S_ISGID** = 1024
**S_ISUID** = 2048
**S_ISVTX** = 512
**S_IWGRP** = 16
**S_IWOTH** = 2
**S_IWRITE** = 128
**S_IWUSR** = 128
**S_IXGRP** = 8
**S_IXOTH** = 1
**S_IXUSR** = 64
**UF_APPEND** = 4
**UF_COMPRESSED** = 32
**UF_HIDDEN** = 32768
**UF_IMMUTABLE** = 2
**UF_NODUMP** = 1
**UF_NOUNLINK** = 16
**UF_OPAQUE** = 8

# bspyclone.bvp

index
d:\work\productioncode\clones\biosppy\bspyclone\bvp__init__.py

```
# import bvp
# import models
```

## Package Contents

bvp

models

# bspyclone.database.h5db

index
d:\work\productioncode\clones\biosppy\bspyclone\database\h5db.py

```
.. module:: h5db
   :platform: Unix, Windows
   :synopsis: This module provides a wrapper to the HDF5 file format, adapting it to store biosignals according to the BioMESH specification at http://cam

.. moduleauthor:: Carlos Carreiras
```

## Modules

h5py  json

## Classes

IOController
hdf
meta
realTime

### class IOController

Methods defined here:

**__init__**(self, fileConfig)

**put**(self, data)

**start**(self)

**stop**(self)

### class hdf

```
Wrapper class to operate on HDF5 records according to the BioMESH specification.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__enter__**(self)

```
__enter__ Method for 'with' statement.

Kwargs:
    None

Kwrvals:
    None

See Also:


Notes:


Example:


References:
    .. [1]
```

**__exit__**(self, exc_type, exc_value, traceback)

```
__exit__ Method for 'with' statement.

Kwargs:
    None

Kwrvals:
    None
```

```
    See Also:

    Notes:

    Example:

    References:
        .. [1]
```

**__init__**(self, filePath=None, mode='a')

```
    Open the HDF5 record.

    Kwargs:
        filePath (str): Path to HDF5 file.

        mode (str): File access mode. Available modes:
            'r+': Read/write, file must exist
            'r': Read only, file must exist
            'w': Create file, truncate if exists
            'w-': Create file, fail if exists
            'a': Read/write if exists, create otherwise
            Default: 'a'.

    Kwrvals:

    See Also:

    Notes:

    Example:
        fid = hdf('record.hdf5', 'a')

    References:
        .. [1]
```

**addEvent**(self, eventType='', timeStamps=None, values=None, mdata=None, eventName=None, compress=False)

```
    Method to add asynchronous data (events) to the HDF5 record.

    Kwargs:
        eventType (str): Type of the events to add. Default: ''.

        timeStamps (array): Array of time stamps. Default: [].

        values (array): Array with data for each time stamp. Default: [].

        mdata (dict): Dictionary object with metadata about the events. Default: {}.

        eventName (str): Name of the group to be created.

        compress (bool): Flag to compress the data (GZIP). Default: False.

    Kwrvals:

    See Also:

    Notes:

    Example:
        fid.addEvent('/test', [0, 1, 2], [[1, 2], [3, 4], [5, 6]], {'comments': 'test event'}, 'event0')

    References:
        .. [1]
```

**addInfo**(self, header={})

```
    Method to add or overwrite the basic information (header) of the HDF5 record.

    Kwargs:
        header (dict): Dictionary (JSON) object with the information. Default: {}.

    Kwrvals:

    See Also:

    Notes:

    Example:
        fid.addInfo({'name': 'record'})

    References:
        .. [1]
```

**addSignal**(self, signalType='', signal=None, mdata=None, dataName=None, compress=False)

```
    Method to add a signal (synchronous data) to the HDF5 record.
```

Kwargs:
signalType (str): Type of the signal to add. Default: ''.

signal (array): Array with the signal to add.

mdata (dict): Dictionary object with metadata about the data. Default: {}.

dataName (str): Name of the dataset to be created.

compress (bool): Flag to compress the data (GZIP). Default: False.

Kwrvals:

See Also:

Notes:

Example:
fid.addSignal('/test', [0, 1, 2], {'comments': '/test signal'}, 'signal0')

References:
.. [1]

**addSignalRT**(self, signalType='', mdata=None, dataName=None, blockShape=None, axis=0, dtype='f8', compress=False)
Method to add a signal (synchronous data) to the HDF5 record in real time.

Kwargs:
signalType (str): Type of the signal to add. Default: ''.

mdata (dict): Dictionary object with metadata about the data. Default: {}.

dataName (str): Name of the dataset to be created.

blockShape (tuple): Shape of the signal blocks.

axis (int): Direction on which the expansion of the dataset is made. Default: 0.

dtype (str): Data type of the signal to add (supports numpy data types).

compress (bool): Flag to compress the data (GZIP). Default: False.

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

**close**(self)
Method to close the HDF5 record.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:
fid.close()

References:
.. [1]

**delEvent**(self, eventType='', eventName=None)
Method to delete asynchronous data (events) from the HDF5 record. The record is marked for repackaging.

Kwargs:
eventType (str): Type of the desired event. Default: ''.

eventName (str): Name of the dataset to retrieve.

Kwrvals:

See Also:

Notes:

```
    Example:
        fid.delEvent('/test', 'event0')

    References:
        .. [1]
```

**delEventType**(self, eventType='')

```
    Method to delete a type of asynchronous data from the HDF5 record. The record is marked for repackaging.

    Kwargs:
        eventType (str): Type of the desired event. Default: ''.

    Kwrvals:


    See Also:


    Notes:


    Example:
        fid.delEventType('/test')

    References:
        .. [1]
```

**delSignal**(self, signalType='', dataName=None)

```
    Method to delete a signal (synchronous data) from the HDF5 record. The record is marked for repackaging.

    Kwargs:
        signalType (str): Type of the desired signal. Default: ''.

        dataName (str): Name of the dataset to retrieve.

    Kwrvals:


    See Also:


    Notes:


    Example:
        fid.delSignal('/test', 'data0')

    References:
        .. [1]
```

**delSignalType**(self, signalType='')

```
    Method to delete a type of signals from the HDF5 record. The record is marked for repackaging.

    Kwargs:
        signalType (str): Signal type to delete. Default: ''.

    Kwrvals:


    See Also:


    Notes:


    Example:
        fid.delSignalType('/test')

    References:
        .. [1]
```

**getEvent**(self, eventType='', eventName=None)

```
    Method to retrieve asynchronous data(events) from the HDF5 record.

    Kwargs:
        eventType (str): Type of the desired event. Default: ''.

        eventName (str): Name of the dataset to retrieve.

    Kwrvals:
        timeStamps (array): Array of time stamps.

        values (array): Array with data for each time stamp.

        mdata (dict): Dictionary object with metadata about the events.

    See Also:


    Notes:


    Example:
        out = fid.getEvent('/test', 'event0')
        timeStamps = out['timeStamps']
```

```
        values = out['values']
        metadata = out['mdata']

    References:
        .. [1]
```

**getEventInfo**(self, eventWeg=None)

```
    Method to retrieve the metadata of asynchronous.

    Kwargs:
        eventWeg (str): Path to the group.

    Kwrvals:
        mdata (dict): Dictionary with the desired information.

    See Also:


    Notes:


    Example:
        out = fid.getEventInfo('/test/event0')
        metadata = out['mdata']

    References:
        .. [1]
```

**getInfo**(self)

```
    Method to retrieve the basic information (header) of the HDF5 record.

    Kwargs:


    Kwrvals:
        header (dict): Dictionary object with the header information.

    See Also:


    Notes:


    Example:
        header = fid.get()['header']

    References:
        .. [1]
```

**getRepack**(self)

```
    Get the repack flag.

    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:
        repack = fid.getRepack()

    References:
        .. [1]
```

**getSignal**(self, signalType='', dataName=None)

```
    Method to retrieve a signal (synchronous data) from the HDF5 record.

    Kwargs:
        signalType (str): Type of the desired signal. Default: ''.

        dataName (str): Name of the dataset to retrieve.

    Kwrvals:
        signal (array): Array with the signals.

        mdata (dict): Dictionary object with metadata about the signals.

    See Also:


    Notes:


    Example:
        out = fid.getSignal('/test', 'signal0')
        signal = out['signal']
        metadata = out['mdata']

    References:
        .. [1]
```

**getSignalInfo**(self, dataWeg=None)

```
Method to retrieve the metadata of a signal.

Kwargs:
    dataWeg (str): Path to the dataset.

Kwrvals:
    mdata (dict): Dictionary with the desired information.

See Also:


Notes:


Example:
    out = fid.getSignalInfo('/test/data0')
    metadata = out['mdata']

References:
    .. [1]
```

**getSignalSet**(self, signalType='', dataName=None)

```
Method to retrieve a signal (synchronous data) from the HDF5 record.

Kwargs:
    signalType (str): Type of the desired signal. Default: ''.

    dataName (str): Name of the dataset to retrieve.

Kwrvals:
    signal (array): Array with the signals.

    mdata (dict): Dictionary object with metadata about the signals.

See Also:


Notes:


Example:
    out = fid.getSignal('/test', 'signal0')
    signal = out['signal']
    metadata = out['mdata']

References:
    .. [1]
```

**listEvents**(self, eventType='')

```
Method to list the events (asynchronous data) belonging to a given type.

Kwargs:
    eventType (str): Type of the desired signal. Default: ''.

Kwrvals:
    eventsList (lit): List of the events of the given type.

See Also:


Notes:


Example:
    fid.listEvents('/test')

References:
    .. [1]
```

**listSignals**(self, signalType='')

```
Method to list the signals (synchronous data) belonging to a given type.

Kwargs:
    signalType (str): Type of the desired signal. Default: ''.

Kwrvals:
    signalsList (lit): List of the signals of the given type.

See Also:


Notes:


Example:
    fid.listSignals('/test')

References:
    .. [1]
```

**setRepack**(self)

```
Set flag to repack.

Kwargs:
```

Kwrvals:

See Also:

Notes:

Example:
fid.setRepack()

References:
.. [1]

**unsetRepack**(self)
Unset flag to repack.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:
fid.unsetRepack()

References:
.. [1]

class **meta**
Wrapper class to store experiments and subjects on HDF5.

Kwargs:

Kwrvals:

See Also:

Notes:

Example:

References:
.. [1]

Methods defined here:

**__enter__**(self)
__enter__ Method for 'with' statement.

Kwargs:
None

Kwrvals:
None

See Also:

Notes:

Example:

References:
.. [1]

**__exit__**(self, exc_type, exc_value, traceback)
__exit__ Method for 'with' statement.

Kwargs:
None

Kwrvals:
None

See Also:

```
    Notes:


    Example:


    References:
        .. [1]
```

**__init__**(self, filePath=None, mode='a')

```
    Open the HDF5 file.

    Kwargs:
        filePath (str): Path to HDF5 file.

        mode (str): File access mode. Available modes:
            'r+': Read/write, file must exist
            'r': Read only, file must exist
            'w': Create file, truncate if exists
            'w-': Create file, fail if exists
            'a': Read/write if exists, create otherwise
            Default: 'a'.

    Kwrvals:


    See Also:


    Notes:


    Example:
        fid = meta('expsub.hdf5', 'a')

    References:
        .. [1]
```

**addExperiment**(self, experiment={})

```
    Method to add an experiment to the file.

    Kwargs:
        experiment (dict): Dictionary with the experiment information. Default: {}.

    Kwrvals:


    See Also:


    Notes:
        The experiment must have a 'name' key.

    Example:
        fid.addExperiment({'name': 'experiment', 'comments': 'Hello world.'})

    References:
        .. [1]
```

**addSubject**(self, subject={})

```
    Method to add a subject to the file.

    Kwargs:
        subject (dict): Dictionary with the subject information. Default: {}.

    Kwrvals:


    See Also:


    Notes:
        The subject must have a '_id' key.

    Example:
        fid.addSubject({'_id': 0, 'name': 'subject'})

    References:
        .. [1]
```

**close**(self)

```
    Method to close the HDF5 file.

    Kwargs:


    Kwrvals:


    See Also:


    Notes:


    Example:
```

```
        fid.close()

    References:
        .. [1]
```

**getExperiment**(self, experimentName=None)

```
    Method to get the information about an experiment.

    Kwargs:
        experimentName (str): The name of the experiment.

    Kwrvals:
        experiment (dict): Dictionary with the experiment information.

    See Also:


    Notes:


    Example:
        experiment = fid.getExperiment('experiment')['experiment']

    References:
        .. [1]
```

**getSubject**(self, subjectId=None)

```
    Method to get the information about a subject.

    Kwargs:
        subjectId (int): The ID of the subject.

    Kwrvals:
        subject (dict): Dictionary with the subject information.

    See Also:


    Notes:


    Example:
        subject = fid.getSubject(0)['subject']

    References:
        .. [1]
```

**listExperiments**(self)

```
    Method to list all the experiments in the file.

    Kwargs:


    Kwrvals:
        expList (list): List with the experiments.

    See Also:


    Notes:


    Example:
        expList = fid.listExperiments()['expList']

    References:
        .. [1]
```

**listSubjects**(self)

```
    Method to list all the subjects in the file.

    Kwargs:


    Kwrvals:
        subList (list): List with the subjects.

    See Also:


    Notes:


    Example:
        subList = fid.listSubjects()['subList']

    References:
        .. [1]
```

**setDB**(self, dbName=None)

```
    Method to set the DB the file belongs to.

    Kwargs:
        dbName (str): Name of the database.

    Kwrvals:
```

```
    See Also:


    Notes:


    Example:
        fid.setDB('database')

    References:
        .. [1]
```

**updateExperiment**(self, experimentName=None, info={})

```
    Method to update an experiment's information.

    Kwargs:
        experimentName (str): Name of the experiment.

        info (dict): Dictionary with the information to update.

    Kwrvals:


    See Also:


    Notes:


    Example:
        fid.updateExperiment('experiment', {'new': 'field'})

    References:
        .. [1]
```

**updateSubject**(self, subjectId=None, info={})

```
    Method to update a subject's information.

    Kwargs:
        sunjectId (int): The ID of the subject.

        info (dict): Dictionary with the information to update.

    Kwrvals:


    See Also:


    Notes:


    Example:
        fid.updateSubject(0, {'new': 'field'})

    References:
        .. [1]
```

class **realTime**

```
Class to add signals in real time to an HDF5 file.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__init__**(self, group, mdata, dataName, blockShape, axis, dtype, compress=False)

```
    Initialize the new dataset.

    Kwargs:
        group (h5py.Group, h5py.File): File or group instance.

        mdata (dict): Dictionary object with metadata about the data.

        dataName (str): Name of the dataset to be created.
```

```
            blockShape (tuple): Shape of the signal blocks.

            axis (int): Direction on which the expansion of the dataset is made.

            dtype (str): Data type of the signal to add (supports numpy data types).

            compress (bool): Flag to compress the data (GZIP). Default: False.

        Kwrvals:


        See Also:


        Notes:


        Example:


        References:
            .. [1]
```

**put**(self, data)

```
        Add data.

        Kwargs:
            data (array): Data to add.

        Kwrvals:


        See Also:


        Notes:


        Example:


        References:
            .. [1]
```

## Functions

**IOProcess**(fileConfig, acquisition, queue)

**latin1ToAscii**(text)

```
    This replaces UNICODE Latin-1 characters with something equivalent in 7-bit ASCII.
    Kwargs:
        text (str): Text to convert.

    Kwrvals:


    See Also:


    Notes:


    Example:
        text = 'João'
        print text
        ntext = latin1ToAscii(text)
        print ntext

    References:
        .. [1] http://stackoverflow.com/questions/930303/python-string-cleanup-manipulation-accented-characters
```

# bspyclone.database

index
d:\work\productioncode\clones\biosppy\bspyclone\database__init__.py

```
# import biomesh
# import h5db
```

## Package Contents

| | | | |
|---|---|---|---|
| HDF5DB | h5db | old2biomesh | repackScript |
| biomesh | h5repack | philipsXML | sandbox |
| example | mongoH5 | physionet | syncdb |

# bspyclone.ecg

index
d:\work\productioncode\clones\biosppy\bspyclone\ecg__init__.py

```
# import ecg
# import models
# import tools
```

## Package Contents

| | | |
|---|---|---|
| ecg | models | models_francis_old |
| hrv | models_francis | tools |

# bspyclone.eda

index
d:\work\productioncode\clones\biosppy\bspyclone\eda__init__.py

```
# import eda
# import models
```

## Package Contents

eda

models

# bspyclone.eeg

index
d:\work\productioncode\clones\biosppy\bspyclone\eeg__init__.py

```
# import eeg
# reload(eeg)
# from eeg import *
```

## Package Contents

eeg

# bspyclone.emg

index
d:\work\productioncode\clones\biosppy\bspyclone\emg__init__.py

```
# import emg
```

## Package Contents

emg

# bspyclone

index
d:\work\productioncode\clones\biosppy\bspyclone__init__.py

BIOSPPY.

.. moduleauthor:: Filipe Canento, Hugo Silva, Andre Lourenco, Prof. Ana Fred

## Package Contents

| | | | |
|---|---|---|---|
| _apiBP | **ecg** (package) | peakd | sandbox |
| apiBP | **eda** (package) | plux | sync |
| bioplux | **eeg** (package) | pluxv2 | timing |
| biopluxwx | **emg** (package) | pywx | tools |
| **bvp** (package) | filt | **resp** (package) | |
| **database** (package) | hr | sand | |

# bspyclone.resp

index
d:\work\productioncode\clones\biosppy\bspyclone\resp__init__.py

```
# import resp
```

## Package Contents

resp