ibpmodel

Release 1.2.1

Ina Rusch

Jun 22, 2023

CONTENTS

The ionospheric bubble probability statistical model is a Swarm L2 product, named IBP_CLI. The output of the Ionospheric Bubble Probability (IBP) product is an index, that depends on the day of year or the month of the year, geographic longitude, local time and solar flux index.

The output floating point index ranges 0-1 and characterizes the percentage probability of low latitude bubble occurence at the specified time, location and solar flux.

This empirical IBP model has been derived from magnetic observations obtained by the CHAMP and Swarm missions. The model is representative for the altitude range 350 - 500 km and low geographic latitudes of +/- 45 degree.

CHAPTER

ONE

CONTENTS

1.1 Installation

To use ibpmodel, first install it using pip:

```
$ pip install ibpmodel
```

ibpmodel has the following dependencies:

- numpy
- pandas
- matplotlib
- scipy (scipy.special)
- cdflib

The required packages are downloaded and installed automatically during the installation.

ibpmodel can also be installed directly from source. The source code can then be downloaded from IAP GitLab and installed:

```
$ git clone https://igit.iap-kborn.de/ibp/ibp-model.git
$ cd ibpmodel
$ pip install .
```

1.2 Usage

1.2.1 Calculation of IBP Index

To calculate the IBP index use *ibpmodel.ibpforward.calculateIBPindex()* function. It returns a pandas.DataFrame:

```
>>> from ibpmodel import calculateIBPindex
>>> calculateIBPindex(day_month=15, longitude=0, local_time=20.9, f107=150)
   Doy  Month  Lon    LT  F10.7     IBP
0   15      1    0  20.9    150  0.3547
```

```
>>> from ibpmodel import calculateIBPindex
>>> calculateIBPindex(day_month=['Jan','Feb','Mar'], local_time=22)
     Doy  Month  Lon  LT  F10.7     IBP
0     15      1 -180  22    150  0.0739
1     15      1 -175  22    150  0.0722
2     15      1 -170  22    150  0.0717
3     15      1 -165  22    150  0.0728
4     15      1 -160  22    150  0.0771
..   ...    ...  ...  ..    ...     ...
211   74      3  155  22    150  0.2061
212   74      3  160  22    150  0.2025
213   74      3  165  22    150  0.1994
214   74      3  170  22    150  0.1967
215   74      3  175  22    150  0.1943

[216 rows x 6 columns]
```

```
>>> from ibpmodel import calculateIBPindex
>>> calculateIBPindex(day_month=[1,15,31], longitude=[-170,175,170], local_time=0,␣
↪f107=120)
   Doy  Month  Lon  LT  F10.7     IBP
0    1      1 -170   0    120  0.0274
1    1      1  175   0    120  0.0304
2    1      1  170   0    120  0.0324
3   15      1 -170   0    120  0.0293
4   15      1  175   0    120  0.0325
5   15      1  170   0    120  0.0347
6   31      1 -170   0    120  0.0341
7   31      1  175   0    120  0.0378
8   31      1  170   0    120  0.0403
```

1.2.2 Read coefficient file

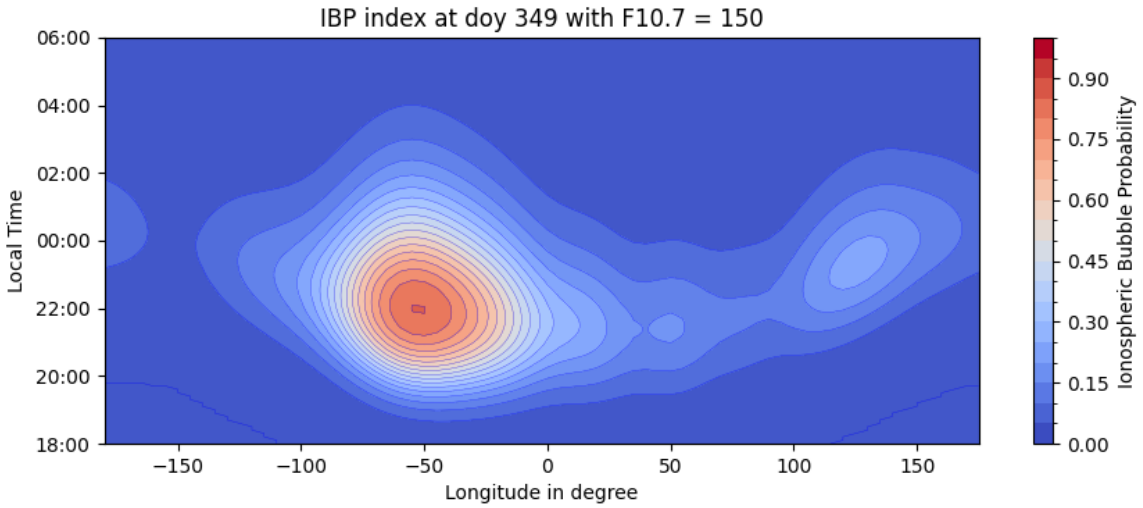
You can load the coefficient file. *ibpmodel.ibpcalc.read_model_file()*:

```
>>> from ibpmodel import read_model_file
>>> c = read_model_file()
>>> c.keys()
dict_keys(['Parameters', 'Intensity', 'Monthly_LT_Shift', 'Density_Estimators', 'Density_
↪Estimator_Lons'])
>>>
>>> c['Intensity']
array([  -6.99518975,   -0.93264132,   96.49049553,    8.81167779,
       -135.50937181])
```

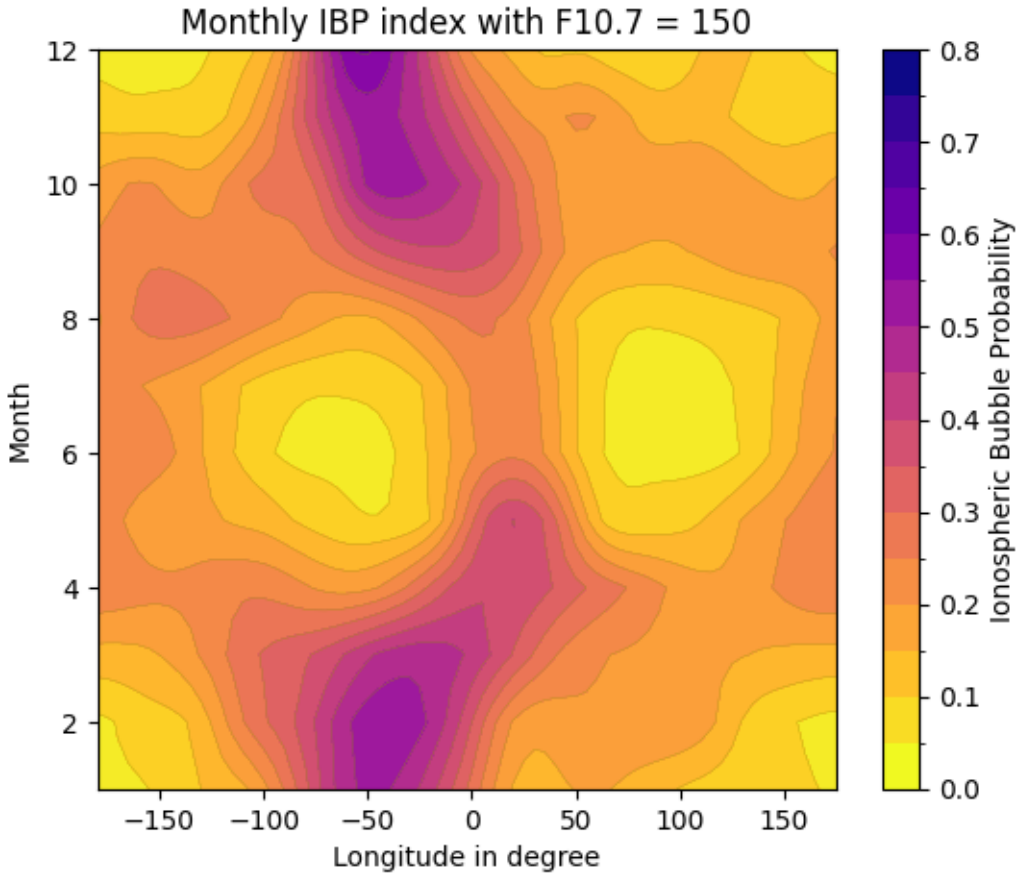
1.2.3 Plotting of the probability

There are two functions to plot IBP index. function *ibpmodel.ibpforward.plotIBPindex()* and *ibpmodel.ibpforward.plotButterflyData()*. By default, the plot is displayed immediately. If you want to make changes or additions, the parameter getFig must be set equal to `True`. Then you get matplat.axis as return value:

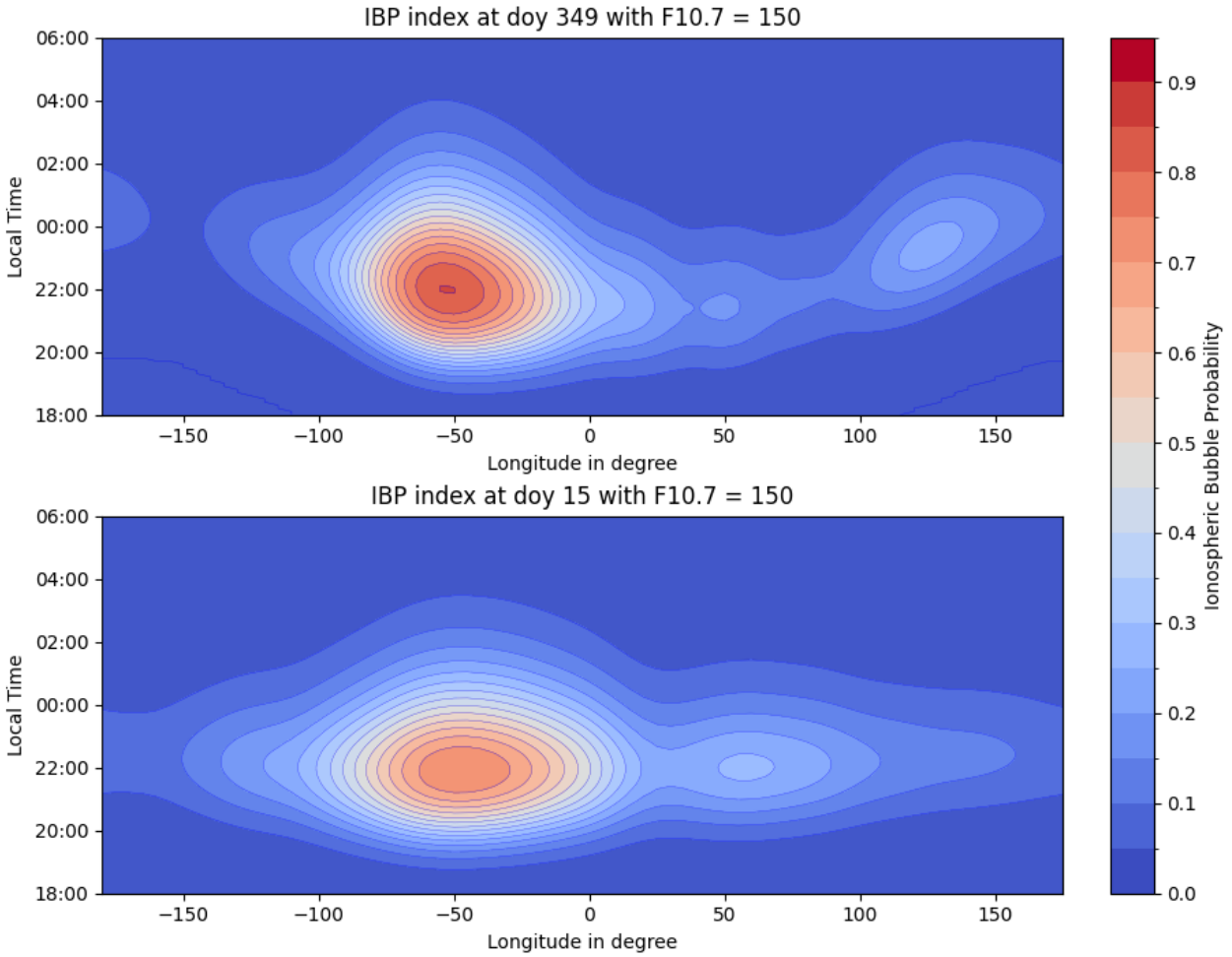
```
>>> import ibpmodel as ibp
>>> ibp.plotIBPindex(doy=349)
```




```
>>> ibp.plotButterflyData(f107=150)
```




```
>>> import ibpmodel as ibp
>>> import matplotlib.pyplot as plt
>>> doys = [349, 15]
>>> fig, axes = plt.subplots(len(doys),1, layout='constrained',figsize=(9, 7))
>>> for d, ax in zip(doys, axes):
...     ax, scalarmap = ibp.plotIBPindex(d, ax=ax)
>>> ibp.ibpforward.setcolorbar(scalarmap, fig, axes, fraction=0.05)
>>> plt.show()
```



1.3 API

This page gives an overview of all functions and methods.

1.3.1 ibpmodel package

ibpmodel.ibpcalc module

This module contains the main functions and the auxiliary/control functions needed to calculate the IBP index.

`ibpmodel.ibpcalc.`**`align_time_of_year`**(*day_of_year*, *month*)

Guess day of year and month from each other.

Parameters

- **day_of_year** (*int or ndarray of ints*) – Time as the day of year, restricted to `0 < day_of_year <= 365`, with the value 0 meaning it should be calculated based on *month* (which gives the median day of the *month*).
- **month** (*int or ndarray of ints*) – Time as the month in the year, with january being month 1, so `0 < month <= 12`.

Returns

- **day_of_year** (*int or ndarray of ints*) – A recalculated possible copy of the *day_of_year* parameter.
- **month** (*int or ndarray of ints*) – A recalculated copy of the *month* parameter.

Examples

```
>>> from ibpmodel import ibpcalc
>>> import numpy as np
>>> ibpcalc.align_time_of_year(0,6)
(166, 6)
>>> ibpcalc.align_time_of_year(162,3)
(162, 6)
>>> ibpcalc.align_time_of_year(0,np.arange(12)+1)
(array([ 16,  45,  75, 105, 136, 166, 197, 228, 258, 289, 319, 350]), array([ 1,  2,
↪  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]))
```

```
>>> ibpcalc.align_time_of_year(np.array([0,33,21,17,267,0,115,96,315,0,172,256]),np.
↪arange(12)+1)
(array([ 16,  33,  21,  17, 267, 166, 115,  96, 315, 289, 172, 256]), array([ 1,  2,
↪  1,  1,  9,  6,  4,  4, 11, 10,  6,  9]))
```

ibpmodel.ibpcalc.**checkDoyMonth**(*day_month*)

Control if input is day of the year (*int*) or abbreviated month name (*str*).

Parameters
day_month (*int or str or list*) – Value to be checked

Returns
doy_out – list of days of the year

Return type
list of int(s)

Example

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.checkDoyMonth(['Mar',200,1])
[74, 200, 1]
```

```
>>> ibpcalc.checkDoyMonth('Dec')
[349]
```

ibpmodel.ibpcalc.**checkParameter**(*para*, *para_range*)

Check if *para* or element of *para* in *para_range*.

Parameters

- **para** (*int or float or list*) – Value to be checked
- **para_range** (*range*) – searching range

Returns
Numpy.array contains *para*.

Return type
numpy.array

Example

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.checkParameter(3,range(0,5))
array([3])
```

```
>>> ibpcalc.checkParameter([0,1,2,3,4],range(0,5))
array([0, 1, 2, 3, 4])
```

ibpmodel.ibpcalc.**compute_lambda**(*longitude*, *params*, *f107*, *gosc_val*, *density*, *month=0*)

Computes the 'lambda' parameter. Lambda is the intensity, one of the parameters describing the final probability and modeled as a Poisson process.

Parameters

- **longitude** (*float or ndarray of floats*) – The longitude(s) of the point(s) we calculate lambda for.
- **params** (*array-like*) – Parameter values from the model (CDF-file).
- **f107** (*float or ndarray of floats*) – The Solar Radio Flux (F10.7 index).
- **gosc_val** (*float or ndarray of floats*) – result of method *fourier_model()*
- **density** (*ndarray of floats*) – Density values from the model (CDF-file).
- **month** (*int or ndarray of ints, optional*) – The number of months since the year started, meaning January would be 0. The default is 0.

Return type
float or ndarray of floats

Examples

```
>>> from ibpmodel import ibpcalc
>>> import numpy as np
>>> ibpcalc.compute_lambda(1,[-232.54229262, 4.67324294], 123.4, 17.3, np.array([2.
↪1,3.0]))
1084.307658528
```

```
>>> params = np.array([-232.54229262, 4.67324294, 1.34695254, -1.31448553, 1.
↪09712955])
>>> densi_once = np.sin(np.arange(360)*np.pi/180) * 3 + 5
>>> ibpcalc.compute_lambda(1,[-232.54229262, 4.67324294], 123.4, 17.3, densi_once)
1788.2556529203105
```

```
>>> densi_year = np.array([ (densi_once -3)* np.cos(month*np.pi/6) + 8 for month in␣
↪range(12)])
>>> ibpcalc.compute_lambda(1,[-232.54229262, 4.67324294], 123.4, 17.3, densi_year,␣
↪month=5)
1851.5944384882494
```

```
>>> months = np.array((np.sin(np.arange(20))+1) % 12, dtype='int')
>>> ibpcalc.compute_lambda(1,[-232.54229262, 4.67324294], 123.4, 17.3, densi_year[:,
↪months], month=months)
array([2149.6915391, 2149.6915391, 2149.6915391, 2149.6915391,
       2149.6915391, 2149.6915391, 2149.6915391, 2149.6915391,
       2149.6915391, 2149.6915391, 2149.6915391, 2149.6915391,
       2149.6915391, 2149.6915391, 2149.6915391, 2149.6915391,
       2149.6915391, 2149.6915391, 2149.6915391, 2149.6915391])
```

ibpmodel.ibpcalc.**compute_probability**(*time*, *expected_bubbles*, *expected_lifetime*, *mu*, *sigma*)

Computes the probability of Ionospheric Plasma Bubbles.

Parameters

- **time** (*float or ndarray of floats*) – The local time, can be eiter as an array or not.
- **expected_bubbles** (*float or ndarray of floats*) – The expected amount of bubbles, can be eiter as an array or not.
- **expected_lifetime** (*float or ndarray of floats*) – The expected lifetime of each bubble, can be eiter as an array or not.
- **mu** (*float or ndarray of floats*) – The mean position of the probability distribution in the model, can be eiter as an array or not.
- **sigma** (*float or ndarray of floats*) – The standard deviation of the probability distribution in the model, can be eiter as an array or not.

Returns

The Ionospheric Bubble Index, value(s) between 0.0 and 1.0, it has the same length as all of the inputs

Return type

float or ndarray of floats

Note: Plasma Bubbles are large-scale depletions compared to the background ionosphere, occurring in the equatorial F-region, in particular after sunset. They are assumably driven by Rayleigh-Taylor instability and already in the past extensively studied by different techniques, showing occurrence probabilities depending on evironmental parameters as season, location, local time and sun activity.

For a climatologic model of these dependencies, extracted from fairly long time series of distortions in the magnetic field readings of the LEO satellites CHAMP (2000-2010) and Swarm (since 2013) the function calculates a probability density.

Examples

```
>>> import numpy as np
>>> from ibpmodel import ibpcalc
>>> ibpcalc.compute_probability(0.0,0.5,0.5,0.5,0.5)
0.049701856965716384
```

```
>>> a = np.arange(24)+0.5
>>> ibpcalc.compute_probability(a,a,a,a,a)
array([0.12259724, 0.32454412, 0.48001002, 0.5996932 , 0.69182957,
```

(continues on next page)

(continued from previous page)

```
       0.76275943, 0.81736377, 0.85940013, 0.89176121, 0.91667393,
       0.93585262, 0.95061706, 0.96198326, 0.97073336, 0.9774695 ,
       0.98265522, 0.98664737, 0.98972067, 0.99208661, 0.99390799,
       0.99531015, 0.99638959, 0.99722058, 0.9978603 ])
```

```
>>> ibpcalc.compute_probability(a,2*a,1.2,1.3,0.4)
array([2.00069488e-02, 7.81272947e-01, 8.55868576e-01, 6.93670227e-01,
       4.83704476e-01, 2.96120013e-01, 1.65026216e-01, 8.64714939e-02,
       4.35684741e-02, 2.14048463e-02, 1.03395302e-02, 4.93490050e-03,
       2.33423700e-03, 1.09629097e-03, 5.11888048e-04, 2.37840684e-04,
       1.10040886e-04, 5.07234746e-05, 2.33043267e-05, 1.06755465e-05,
       4.87751438e-06, 2.22316484e-06, 1.01112283e-06, 4.58962618e-07])
```

ibpmodel.ibpcalc.**compute_probability_exp**(*day_of_year*, *month*, *longitude*, *local_time*, *f107*, *data*)

Compute the Ionospheric Bubble index. Core routine to return a probability of the occurence of a bubble.

Parameters

- **day_of_year** (*int or ndarray of ints*) – Time as the day of year, restricted to `0 < day_of_year <= 356`, with the value 0 meaning it should be calculated based on month (which gives the median day of the month).
- **month** (*int or ndarray of ints*) – Time as the month in the year, with january being month 1, `0 < month <= 12`.
- **longitude** (*float or ndarray of floats*) – The geographical longitude(s), `-180 <= longitude <= 180`.
- **local_time** (*float or ndarray of floats*) – The local time, can be eiter as an array or not, `-6.0 <= local_time <= 24`.
- **f107** (*float or ndarray of floats*) – The Solar Radio Flux (F10.7 index), `0.0 <= f107 <= 200.0`.
- **data** (*dict*) – Containing the parameters of the model (CDF-file).

Returns

The Ionospheric Bubble Index, value(s) between 0.0 and 1.0, it has the same length as all of the inputs

Return type

float or ndarray of floats

Examples

```
>>> from ibpmodel import ibpcalc
>>> import numpy as np
>>> data = ibpcalc.read_model_file()
>>> ibpcalc.compute_probability_exp(0, 3, 12, 2, 150, data)
0.04537450559812162
```

```
>>> ibpcalc.compute_probability_exp(0, 2, 12, 2, 150, data)
0.03213949936284399
```

Note: Rearranged/rewritten and optimized by Ask Neve Gamby <aknvg@space.dtu.dk>.

The resolution of the function *gosc* is higher than monthly. If a *day_of_year* is known, the results can be more precise.

It is now possible to calculate with either ndarrays or single values, for all parameters except data (which has a special structure). Note that this version is, compared with the initial version more closely resembling the original 'R' code, much more efficient due to vectorization.

ibpmodel.ibpcalc.**doyFromMonth**(*month*)

Calculate day of year from the 15th of the month

Parameters
month (*int*) – Value as the month in the year, with january being month 1, 1 <= month <= 12

Returns
Day of year.

Return type
int

Example

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.doyFromMonth(7)
196
```

ibpmodel.ibpcalc.**fourier_model**(*coeffients*, *theta*, *periode=365.0*)

Computes a value based on a model of fourier components up to 2nd degree.

Parameters

- **coeffients** (*array-like of numbers*) – The coefficients of the fourier series up to second degree, with even numbers representing cosinus and odd sinus. The shape of *coefficients* must be *(5,*theta.shape)*.
- **theta** (*number or ndarray of numbers*) – The part representing the phase in the fourier expansion. It is in the same units as the periode.
- **periode** (*number, optional*) – The amount of *theta* need for one periode of the 1st degree of the fourier expansion. Defaults to 365 (the days in a year).

Returns
The shape of this is equivalent to the shape of *theta*.

Return type
number or ndarray of numbers

Examples

```
>>> from ibpmodel import ibpcalc
>>> import numpy as np
>>> ibpcalc.fourier_model([0,1,-1,0,0],180)
1.0420963481067607
```

```
>>> ibpcalc.fourier_model([0,1,-1,-0.5,0.7],1,periode=10)
-0.48044810416758793
```

```
>>> ibpcalc.fourier_model([0,1,-1,-0.5,0.7],np.arange(11),10)
array([-0.3       , -0.4804481 , -0.218165  ,  0.98765424,  2.0886424 ,
        1.7       , -0.03798462, -1.50224404, -1.53249278, -0.70496209,
       -0.3       ])
```

ibpmodel.ibpcalc.**monthFromDoy**(*doy*)

Calculate month from day of the year

Parameters
doy (*int*) – Day of the year, `1 <= doy <= 365`.

Returns
Value as the month in the year, with january being month 1.

Return type
int

Example

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.monthFromDoy(275)
10
```

ibpmodel.ibpcalc.**monthfromString**(*month_str*)

Convert abbreviated month name to *int*

Parameters
month_str (*str*) – Abbreviated month name. *['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']*

Returns
Value as the month in the year, with january being month 1.

Return type
int

Example

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.monthfromString('Mar')
3
```

ibpmodel.ibpcalc.**read_model_file**(*file=None*)

Load CDF content into a dictionary. If no file is declared, the CDF file included in the package will be used. (SW_OPER_IBP_CLI_2__00000000T000000_99999999T999999_0002.cdf)

Parameters
file (*file path, optional*) – Path to cdf file containing parametors for the model. The default is None.

Returns
Contains the content of the CDF file.

Return type
dict

ibpmodel.ibpcalc.**tile_aggregate**(*result*, **args*, *aggregator=<function mean>*)

Compresses tiles and aggregate results on the last axis of tiled ranges

Parameters

- **result** (*numpy.array*) – Array that is aggregated.
- ***args** (*list or numpy.array*) – Along the last element is crompressed.
- **aggregator** (*function, optional*) – Specifies how to aggregate. the defaults is *np.mean*.

Returns
last element is the compressed result

Return type
list of numpy.array

Example

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.tile_aggregate(np.array([3,2,3,3,2,1,2,1]), [10,12], [20,21,22,24])
(10, 12, array([2.75, 1.5 ]))
```

```
>>> ibpcalc.tile_aggregate(np.array([3,2]), np.array([12]), np.array([20,21]))
(12, array([2.5]))
```

```
>>> ibpcalc.tile_aggregate(np.array([3,2,2,1,4,2,3,3,1,4,8,5,9,6,7,4,5,2,1,6,7,9,5,
↪7]), [9,10,11], [20,21], [7,8,5,6])
(array([ 9,  9, 10, 10, 11, 11]), array([20, 21, 20, 21, 20, 21]), array([2. , 3. ,␣
↪4.5, 6.5, 3.5, 7. ]))
```

ibpmodel.ibpcalc.**tiler**(**args*)

Provides mixed combinations of arguments.

Creates copies of the arguments that have length equal to the product of the length of the arguments. This results in an ordered set of all combinations of the individual values from each of the arguements.

Parameters
***args** (*array-likes*) – Each argument is an array-like of possibly different length.

Returns
A list of ordered combination of the input arguments.

Return type
list of array-likes

Examples

```
>>> from ibpmodel import ibpcalc
>>> ibpcalc.tiler([1, 2, 3],['A', 'B'])
[array([1, 1, 2, 2, 3, 3]), array(['A', 'B', 'A', 'B', 'A', 'B'], dtype='<U1')]
```

```
>>> ibpcalc.tiler([17,13],[1, 2, 3],['A', 'B'])
[array([17, 17, 17, 17, 17, 17, 13, 13, 13, 13, 13, 13]), array([1, 1, 2, 2, 3, 3,␣
↪1, 1, 2, 2, 3, 3]), array(['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A',
↪'B'],
      dtype='<U1')]
```

ibpmodel.ibpforward module

This module contains the application functions for the calculation of the IBP index as well as the graphical visualization.

ibpmodel.ibpforward.**butterflyData**(*f107=150*, *coeff=None*)

Calculates the Ionospheric Bubble Propability Index for all months (using the center DOY of each month) and all integer longitudes (resolution of 5 degree) using Local_Time of range -5 to 1 and a fixed value of F10.7. IBP index is then averaged from the Local_Times.

Parameters
f107 (*int, optional*) – The Solar Radio Flux (F10.7 index). The default is 150.

Returns
out_data – [[month],[longitude],[IBP index]].

Return type
numpy.array

ibpmodel.ibpforward.**calculateIBPindex**(*day_month=0*, *longitude=[-180, -175, -170, -165, -160, -155, -150, -145, -140, -135, -130, -125, -120, -115, -110, -105, -100, -95, -90, -85, -80, -75, -70, -65, -60, -55, -50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175]*, *local_time=array([-6.00000000e+00, -5.90000000e+00, -5.80000000e+00, -5.70000000e+00, -5.60000000e+00, -5.50000000e+00, -5.40000000e+00, -5.30000000e+00, -5.20000000e+00, -5.10000000e+00, -5.00000000e+00, -4.90000000e+00, -4.80000000e+00, -4.70000000e+00, -4.60000000e+00, -4.50000000e+00, -4.40000000e+00, -4.30000000e+00, -4.20000000e+00, -4.10000000e+00, -4.00000000e+00, -3.90000000e+00, -3.80000000e+00, -3.70000000e+00, -3.60000000e+00, -3.50000000e+00, -3.40000000e+00, -3.30000000e+00, -3.20000000e+00, -3.10000000e+00, -3.00000000e+00, -2.90000000e+00, -2.80000000e+00, -2.70000000e+00, -2.60000000e+00, -2.50000000e+00, -2.40000000e+00, -2.30000000e+00, -2.20000000e+00, -2.10000000e+00, -2.00000000e+00, -1.90000000e+00, -1.80000000e+00, -1.70000000e+00, -1.60000000e+00, -1.50000000e+00, -1.40000000e+00, -1.30000000e+00, -1.20000000e+00, -1.10000000e+00, -1.00000000e+00, -9.00000000e-01, -8.00000000e-01, -7.00000000e-01, -6.00000000e-01, -5.00000000e-01, -4.00000000e-01, -3.00000000e-01, -2.00000000e-01, -1.00000000e-01, -2.13162821e-14, 1.00000000e-01, 2.00000000e-01, 3.00000000e-01, 4.00000000e-01, 5.00000000e-01, 6.00000000e-01, 7.00000000e-01, 8.00000000e-01, 9.00000000e-01, 1.00000000e+00, 1.10000000e+00, 1.20000000e+00, 1.30000000e+00, 1.40000000e+00, 1.50000000e+00, 1.60000000e+00, 1.70000000e+00, 1.80000000e+00, 1.90000000e+00, 2.00000000e+00, 2.10000000e+00, 2.20000000e+00, 2.30000000e+00, 2.40000000e+00, 2.50000000e+00, 2.60000000e+00, 2.70000000e+00, 2.80000000e+00, 2.90000000e+00, 3.00000000e+00, 3.10000000e+00, 3.20000000e+00, 3.30000000e+00, 3.40000000e+00, 3.50000000e+00, 3.60000000e+00, 3.70000000e+00, 3.80000000e+00, 3.90000000e+00, 4.00000000e+00, 4.10000000e+00, 4.20000000e+00, 4.30000000e+00, 4.40000000e+00, 4.50000000e+00, 4.60000000e+00, 4.70000000e+00, 4.80000000e+00, 4.90000000e+00, 5.00000000e+00, 5.10000000e+00, 5.20000000e+00, 5.30000000e+00, 5.40000000e+00, 5.50000000e+00, 5.60000000e+00, 5.70000000e+00, 5.80000000e+00, 5.90000000e+00, 6.00000000e+00])*, *f107=150*, *coeff=None*)

Calculates the Ionospheric Bubble propability index based on the input parameters. Returns a *pandas.DataFrame* with input parameters and IBP index.

Parameters

- **day_month** (*int or str or list, optional*) – Day of year (*int*) or the month of the year (*str*). *int*: Day of the year, `0 <= doy <= 365`. The value 0 means it should be calculated based on every month. *str*: Abbreviated month name. [‘Jan’, ‘Feb’, ‘Mar’, ‘Apr’,

'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'] The default is 0.

- **longitude** (*int or list of ints, optional*) – The geographical longitude(s), `-180 <= longitude <= 180`. The default is *list(range(-180,180,5))*.
- **local_time** (*int or float or list, optional*) – The local time, `-6.0 <= local_time <= 24`. The default is *np.arange(-6,6.1,0.1)*.
- **f107** (*int or float or list, optional*) – The Solar Radio Flux (F10.7 index), `80.0 <= f107`. The default is 150.
- **coeff** (*None or str, optional*) – Name of the coefficient file. The default is None.

Returns
df – contains the columns: Doy (Day(s) of the year), Month (Month(s) from the day of the year), Lon (Longitude(s)), LT (Local Time(s)), F10.7 (solar index(es)), IBP (Ionospheric Bubble Index, value(s) between 0.0 and 1.0).

Return type
pandas.DataFrame

ibpmodel.ibpforward.**checkcmap**(*cmap*)

Check if variable is a color map.

Parameters
cmap (*str or matplotlib.colors.Colormap*) –

Return type
matplotlib.colors.Colormap

ibpmodel.ibpforward.**getcolorbar**(*cmap*, *level=array([0., 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.])*)

Applies level to colormap.

Parameters

- **cmap** (*matplotlib.colors.Colormap*) –
- **level** (*array-like*) –

Return type
matplotlib.cm.ScalarMappable

ibpmodel.ibpforward.**plotButterflyData**(*f107=150*, *ax=None*, *coeff=None*, *cmap='plasma_r'*, *colors='#999900'*, *linewidths=0.2*, ***kwargs*)

Create a contour plot of the result from function butterflyData(). Default colormap is 'plasma_r'.

Parameters

- **f107** (*int or float, optional*) – The Solar Radio Flux (F10.7 index). The default is 150.
- **ax** (*matplotlib.axes, optional*) – The Axes object in which the plot will be drawn. The default is None.
- **coeff** (*str, optional*) – Path of coefficient file. The default is None.
- **cmap** (*str or Colormap, optional*) – The colormap instance or registered colormap name to use. The default is 'plasma_r'.
- **colors** (*color string or sequence of colors, optional*) – The color of the contour lines. The default is '#999900'.
- **linewidths** (*float, optional*) – The line width of the contour lines. The default is 0.2.

Return type
matplotlib.axes, matplotlib.cm.ScalarMappable

ibpmodel.ibpforward.**plotIBPindex**(*doy*, *f107=150*, *ax=None*, *coeff=None*, *cmap='coolwarm'*, *colors='b'*, *linewidths=0.2*, ***kwargs*)

Create a contour plot of IBP index for the given day. The resolution along the longitude is 5 degree. Local time spans from 6 pm to 6 am with a resolution of 0.1 hours. Default colormap is 'coolwarm'.

Parameters

- **doy** (*int or str*) – Day of year (*int*) or the month of the year (*str*). *int*: Day of the year, `0 <= doy <= 365`. The value 0 means it should be calculated based on every month. *str*: Abbreviated month name. ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
- **f107** (*int or float, optional*) – The Solar Radio Flux (F10.7 index). The default is 150.
- **ax** (*matplotlib.axes, optional*) – The Axes object in which the plot will be drawn. The default is None.
- **coeff** (*str, optional*) – Path of coefficient file. The default is None.
- **cmap** (*str or Colormap, optional*) – The colormap instance or registered colormap name to use. The default is 'coolwarm'.
- **colors** (*color string or sequence of colors, optional*) – The color of the contour lines. The default is 'blue'.
- **linewidths** (*float, optional*) – The line width of the contour lines. The default is 0.2.

Return type
matplotlib.axes, matplotlib.cm.ScalarMappable

ibpmodel.ibpforward.**setcolorbar**(*scalarmap*, *fig*, *ax*, ***kwargs*)

Sets colorbar to figure on the specified axis.

Parameters

- **scalarmap** (*matplotlib.cm.ScalarMappable*) –
- **fig** (*matplotlib.figure.Figure*) –
- **ax** (*matplotlib.axes*) –

Return type
colorbar

1.4 References

Stolle et al., An empirical climatological model of the occurrence of F region equatorial plasma irregularities, 8th Swarm data quality workshop at ESA/ESRIN, October 2017.

Lucas Schreiter, Anwendungsorientierte Modellierung der Auftretenswahrscheinlichkeit und relativen Häufigkeit von äquatorialen Plasmabubbles, Master's thesis, Institute of Mathematics, University of Potsdam, 2016. (in German only.)

1.5 Changelog

Version 1.0.0: Adding a module with application functions for the calculation of the IBP index as well as the graphical visualization.
Version 1.0.1: Create Sphinx documentation.
Version 1.0.2: Rebuild package structure.
Version 1.0.3: Integration into Read the Docs (https://ibp-model.readthedocs.io)
Version 1.0.4: Move from https://test.pypi.org to https://pypi.org
Version 1.1.0: Update of the coefficient file (SW_OPER_IBP_CLI_2__00000000T000000_99999999T999999_0002.cdf)
Version 1.1.1: Correction at Sphinx setting
Version 1.1.2: Change package init settings
Version 1.2.0: Adjustments to the plot functions
Version 1.2.1: Limitation of Solar Radio Flux and output of a warning message; Partitioning requirements.txt

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX