pythermalcomfort

Release 2.10.0

Federico Tartarini

Mar 18, 2024

CONTENTS

CHAPTER ONE

OVERVIEW

docs	
license	
downloads	
tests	
package	

Package to calculate several thermal comfort indices (e.g. PMV, PPD, SET, adaptive) and convert physical variables.

Please cite us if you use this package: Tartarini, F., Schiavon, S., 2020. pythermalcomfort: A Python package for thermal comfort research. SoftwareX 12, 100578. https://doi.org/10.1016/j.softx.2020.100578

- Free software: MIT license

1.1 Installation

```
pip install pythermalcomfort
```

You can also install the in-development version with:

```
pip install https://github.com/CenterForTheBuiltEnvironment/pythermalcomfort/archive/
↪master.zip
```

1.2 Documentation

https://pythermalcomfort.readthedocs.io/

1.3 Examples and Tutorials

Examples files on how to use some of the functions

1.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. Click here to learn more on how to contribute to the project.

1.5 Deployment

I am using travis to test the code. In addition, I have enabled GitHub actions. Every time the code is pushed or pulled to the *master* repository then the GitHub action tests the code and if the tests pass, a new version of the package is published automatically on PyPI. See file in *.github/workflows/* for more information.

CHAPTER TWO

INSTALLATION

At the command line:

```
pip install pythermalcomfort
```

CHAPTER

THREE

EXAMPLES AND TUTORIALS

Examples files on how to use some of the functions

YouTube tutorials playlist

CHAPTER

FOUR

FUNCTIONS DOCUMENTATION

4.1 Comfort models

4.1.1 Adaptive ASHRAE

`pythermalcomfort.models.adaptive_ashrae.`**`adaptive_ashrae`**(*tdb*, *tr*, *t_running_mean*, *v*, *units='SI'*, *limit_inputs=True*)

Determines the adaptive thermal comfort based on ASHRAE 55. The adaptive model relates indoor design temperatures or acceptable temperature ranges to outdoor meteorological or climatological parameters. The adaptive model can only be used in occupant-controlled naturally conditioned spaces that meet all the following criteria:

- There is no mechianical cooling or heating system in operation
- Occupants have a metabolic rate between 1.0 and 1.5 met
- Occupants are free to adapt their clothing within a range as wide as 0.5 and 1.0 clo
- The prevailing mean (runnin mean) outdoor temperature is between 10 and 33.5 °C

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **t_running_mean** (*float, int, or array-like*) – running mean temperature, default in [°C] in [°C] in [°F] if *units* = 'IP'

  The running mean temperature can be calculated using the function `pythermalcomfort.utilities.running_mean_outdoor_temperature()`.
- **v** (*float, int, or array-like*) – air speed, default in [m/s] in [fps] if *units* = 'IP'
- **units** (*str, optional*) – select the SI (International System of Units) or the IP (Imperial Units) system. Supported values are 'SI' and 'IP'. Defaults to 'SI'.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

  The ASHRAE 55 2020 limits are 10 < tdb [°C] < 40, 10 < tr [°C] < 40, 0 < vr [m/s] < 2, 10 < t running mean [°C] < 33.5

Returns

- **tmp_cmf** (*float, int, or array-like*) – Comfort temperature a that specific running mean temperature, default in [°C] or in [°F]
- **tmp_cmf_80_low** (*float, int, or array-like*) – Lower acceptable comfort temperature for 80% occupants, default in [°C] or in [°F]
- **tmp_cmf_80_up** (*float, int, or array-like*) – Upper acceptable comfort temperature for 80% occupants, default in [°C] or in [°F]
- **tmp_cmf_90_low** (*float, int, or array-like*) – Lower acceptable comfort temperature for 90% occupants, default in [°C] or in [°F]
- **tmp_cmf_90_up** (*float, int, or array-like*) – Upper acceptable comfort temperature for 90% occupants, default in [°C] or in [°F]
- **acceptability_80** (*bol or array-like*) – Acceptability for 80% occupants
- **acceptability_90** (*bol or array-like*) – Acceptability for 90% occupants

Notes

You can use this function to calculate if your conditions are within the *adaptive thermal comfort region*. Calculations with comply with the ASHRAE 55 2020 Standard[1].

Examples

```
>>> from pythermalcomfort.models import adaptive_ashrae
>>> results = adaptive_ashrae(tdb=25, tr=25, t_running_mean=20, v=0.1)
>>> print(results)
{'tmp_cmf': 24.0, 'tmp_cmf_80_low': 20.5, 'tmp_cmf_80_up': 27.5,
'tmp_cmf_90_low': 21.5, 'tmp_cmf_90_up': 26.5, 'acceptability_80': array(True),
'acceptability_90': array(True)}

>>> print(results.acceptability_80)  # or use print(results["acceptability_80"])
True
# The conditions you entered are considered to be comfortable for by 80% of the
occupants

>>> # You can also pass arrays as input to the function
>>> results = adaptive_ashrae(tdb=[25, 26], tr=25, t_running_mean=20, v=0.1)
>>> print(results)
{'tmp_cmf': 24.0, 'tmp_cmf_80_low': 20.5, 'tmp_cmf_80_up': 27.5,
'tmp_cmf_90_low': 21.5, 'tmp_cmf_90_up': 26.5, 'acceptability_80': array(True),
'acceptability_90': array(True)}

>>> # for users who want to use the IP system
>>> results = adaptive_ashrae(tdb=77, tr=77, t_running_mean=68, v=0.3, units='ip')
>>> print(results)
{'tmp_cmf': 75.2, 'tmp_cmf_80_low': 68.9, 'tmp_cmf_80_up': 81.5,
'tmp_cmf_90_low': 70.7, 'tmp_cmf_90_up': 79.7, 'acceptability_80': array(True),
'acceptability_90': array(True)}

>>> adaptive_ashrae(tdb=25, tr=25, t_running_mean=9, v=0.1)
```

(continues on next page)

[1] ANSI, & ASHRAE. (2020). Thermal Environmental Conditions for Human Occupancy. Atlanta.

(continued from previous page)

```
{'tmp_cmf': nan, 'tmp_cmf_80_low': nan, ... }
# The adaptive thermal comfort model can only be used
# if the running mean temperature is higher than 10°C
```

4.1.2 Adaptive EN

`pythermalcomfort.models.adaptive_en.`**`adaptive_en`**(*tdb*, *tr*, *t_running_mean*, *v*, *units='SI'*, *limit_inputs=True*)

Determines the adaptive thermal comfort based on EN 16798-1 2019[3]

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **t_running_mean** (*float, int, or array-like*) – running mean temperature, default in [°C] in [°C] in [°F] if *units* = 'IP'

  The running mean temperature can be calculated using the function `pythermalcomfort.utilities.running_mean_outdoor_temperature()`.
- **v** (*float, int, or array-like*) – air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: Indoor operative temperature correction is applicable for buildings equipped with fans or personal systems providing building occupants with personal control over air speed at occupant level. For operative temperatures above 25°C the comfort zone upper limit can be increased by 1.2 °C ($0.6 < v < 0.9$ m/s), 1.8 °C ($0.9 < v < 1.2$ m/s), 2.2 °C ($v > 1.2$ m/s)
- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

Returns

- **tmp_cmf** (*float, int, or array-like*) – Comfort temperature at that specific running mean temperature, default in [°C] or in [°F]
- **acceptability_cat_i** (*bol or array-like*) – If the indoor conditions comply with comfort category I
- **acceptability_cat_ii** (*bol or array-like*) – If the indoor conditions comply with comfort category II
- **acceptability_cat_iii** (*bol or array-like*) – If the indoor conditions comply with comfort category III
- **tmp_cmf_cat_i_up** (*float, int, or array-like*) – Upper acceptable comfort temperature for category I, default in [°C] or in [°F]
- **tmp_cmf_cat_ii_up** (*float, int, or array-like*) – Upper acceptable comfort temperature for category II, default in [°C] or in [°F]
- **tmp_cmf_cat_iii_up** (*float, int, or array-like*) – Upper acceptable comfort temperature for category III, default in [°C] or in [°F]

[3] EN, & BSI. (2019). Energy performance of buildings - Ventilation for buildings. BSI Standards Limited 2019.

- **tmp_cmf_cat_i_low** (*float, int, or array-like*) – Lower acceptable comfort temperature for category I, default in [°C] or in [°F]
- **tmp_cmf_cat_ii_low** (*float, int, or array-like*) – Lower acceptable comfort temperature for category II, default in [°C] or in [°F]
- **tmp_cmf_cat_iii_low** (*float or array-like*) – Lower acceptable comfort temperature for category III, default in [°C] or in [°F]

Notes

You can use this function to calculate if your conditions are within the EN adaptive thermal comfort region. Calculations with comply with the EN 16798-1 2019[Page 9, 3].

Examples

```
>>> from pythermalcomfort.models import adaptive_en
>>> results = adaptive_en(tdb=25, tr=25, t_running_mean=20, v=0.1)
>>> print(results)
{'tmp_cmf': 25.4, 'acceptability_cat_i': True, 'acceptability_cat_ii': True, ... }

>>> print(results['acceptability_cat_i'])
True
# The conditions you entered are considered to comply with Category I

>>> # for users who wants to use the IP system
>>> results = adaptive_en(tdb=77, tr=77, t_running_mean=68, v=0.3, units='ip')
>>> print(results)
{'tmp_cmf': 77.7, 'acceptability_cat_i': True, 'acceptability_cat_ii': True, ... }

>>> results = adaptive_en(tdb=25, tr=25, t_running_mean=9, v=0.1)
{'tmp_cmf': nan, 'acceptability_cat_i': True, 'acceptability_cat_ii': True, ... }
# The adaptive thermal comfort model can only be used
# if the running mean temperature is between 10 °C and 30 °C
```

4.1.3 Adaptive Predicted Mean Vote (aPMV)

`pythermalcomfort.models.a_pmv.`**`a_pmv`**(*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *a_coefficient*, *wme=0*, *units='SI'*, *limit_inputs=True*)

Returns Adaptive Predicted Mean Vote (aPMV)[25]. This index was developed by Yao, R. et al. (2009). The model takes into account factors such as culture, climate, social, psychological and behavioural adaptations, which have an impact on the senses used to detect thermal comfort. This model uses an adaptive coefficient () representing the adaptive factors that affect the sense of thermal comfort.

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'

[25] Yao, Runming & Li, Baizhan & Liu, Jing. (2009). A theoretical adaptive model of thermal comfort – Adaptive Predicted Mean Vote (aPMV). Building and Environment. 44. 2089-2096. 10.1016/j.buildenv.2009.02.014.

- **vr** (*float, int, or array-like*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function `pythermalcomfort.utilities.v_relative()`.

- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **met** (*float, int, or array-like*) – metabolic rate, [met]
- **clo** (*float, int, or array-like*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently, the ISO 7730 states that the clothing insulation shall be corrected[2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function `pythermalcomfort.utilities.clo_dynamic()`.

- **a_coefficient** (*float*) – adaptive coefficient
- **wme** (*float, int, or array-like*) – external work, [met] default 0
- **units** (*str, optional*) – select the SI (International System of Units) or the IP (Imperial Units) system. Supported values are 'SI' and 'IP'. Defaults to 'SI'.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

  The ISO 7730 2005 limits are 10 < tdb [°C] < 30, 10 < tr [°C] < 40, 0 < vr [m/s] < 1, 0.8 < met [met] < 4, 0 < clo [clo] < 2, and -2 < PMV < 2.

Returns
pmv (*float, int, or array-like*) – Predicted Mean Vote

Examples

```
>>> from pythermalcomfort.models import a_pmv
>>> from pythermalcomfort.utilities import v_relative, clo_dynamic
>>> v = 0.1
>>> met = 1.4
>>> clo = 0.5
>>> # calculate relative air speed
>>> v_r = v_relative(v=v, met=met)
>>> # calculate dynamic clothing
>>> clo_d = clo_dynamic(clo=clo, met=met)
>>> results = a_pmv(tdb=28, tr=28, vr=v_r, rh=50, met=met, clo=clo_d, a_
↪coefficient=0.293)
>>> print(results)
0.74
```

[2] ISO. (2005). ISO 7730 - Ergonomics of the thermal environment — Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria.

4.1.4 Adaptive Thermal Heat Balance (ATHB)

`pythermalcomfort.models.athb.athb`(*tdb*, *tr*, *vr*, *rh*, *met*, *t_running_mean*)

Return the PMV value calculated with the Adaptive Thermal Heat Balance Framework[27]. The adaptive thermal heat balance (ATHB) framework introduced a method to account for the three adaptive principals, namely physiological, behavioral, and psychological adaptation, individually within existing heat balance models. The objective is a predictive model of thermal sensation applicable during the design stage or in international standards without knowing characteristics of future occupants.

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, in [°C]
- **tr** (*float, int, or array-like*) – mean radiant temperature, in [°C]
- **vr** (*float, int, or array-like*) – relative air speed, in [m/s]

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function *pythermalcomfort.utilities.v_relative()*.
- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **met** (*float, int, or array-like*) – metabolic rate, [met]
- **t_running_mean** (*float or array-like*) – running mean temperature, in [°C]

  The running mean temperature can be calculated using the function *pythermalcomfort.utilities.running_mean_outdoor_temperature()*.

Returns

athb_pmv (*float or array-like*) – Predicted Mean Vote calculated with the Adaptive Thermal Heat Balance framework

Examples

```
>>> from pythermalcomfort.models import athb
>>> print(athb( tdb=[25, 27], tr=25, vr=0.1, rh=50, met=1.1, t_running_mean=20))
[0.2, 0.209]
```

4.1.5 Adjusted Predicted Mean Votes with Expectancy Factor (ePMV)

`pythermalcomfort.models.e_pmv.e_pmv`(*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *e_coefficient*, *wme=0*, ***kwargs*)

Returns Adjusted Predicted Mean Votes with Expectancy Factor (ePMV). This index was developed by Fanger, P. O. et al. (2002). In non-air- conditioned buildings in warm climates, occupants may sense the warmth as being less severe than the PMV predicts. The main reason is low expectations, but a metabolic rate that is estimated too high can also contribute to explaining the difference. An extension of the PMV model that includes an expectancy factor is introduced for use in non-air-conditioned buildings in warm climates[26].

Parameters

[27] Schweiker, M., 2022. Combining adaptive and heat balance models for thermal sensation prediction: A new approach towards a theory and data-driven adaptive thermal heat balance model. Indoor Air 32, 1–19. DOI: doi.org/10.1111/ina.13018

[26] Fanger, P. & Toftum, Jorn. (2002). Extension of the PMV model to non-air-conditioned buildings in warm climates. Energy and Buildings. 34. 533-536. 10.1016/S0378-7788(02)00003-8.

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **vr** (*float, int, or array-like*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function `pythermalcomfort.utilities.v_relative()`.
- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **met** (*float, int, or array-like*) – metabolic rate, [met]
- **clo** (*float, int, or array-like*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently, the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function `pythermalcomfort.utilities.clo_dynamic()`.
- **e_coefficient** (*float*) – expectacy factor
- **wme** (*float, int, or array-like*) – external work, [met] default 0

Other Parameters

- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

  The ISO 7730 2005 limits are 10 < tdb [°C] < 30, 10 < tr [°C] < 40, 0 < vr [m/s] < 1, 0.8 < met [met] < 4, 0 < clo [clo] < 2, and -2 < PMV < 2.

Returns

pmv (*float, int, or array-like*) – Predicted Mean Vote

Examples

```
>>> from pythermalcomfort.models import a_pmv
>>> from pythermalcomfort.utilities import v_relative, clo_dynamic
>>> tdb = 28
>>> tr = 28
>>> rh = 50
>>> v = 0.1
>>> met = 1.4
>>> clo = 0.5
>>> # calculate relative air speed
>>> v_r = v_relative(v=v, met=met)
>>> # calculate dynamic clothing
>>> clo_d = clo_dynamic(clo=clo, met=met)
>>> results = e_pmv(tdb, tr, v_r, rh, met, clo_d, e_coefficient=0.6)
```

(continues on next page)

(continued from previous page)

```
>>> print(results)
0.51
```

4.1.6 Apparent Temperature (AT)

`pythermalcomfort.models.at.at`(*tdb*, *rh*, *v*, *q=None*, ***kwargs*)

Calculates the Apparent Temperature (AT). The AT is defined as the temperature at the reference humidity level producing the same amount of discomfort as that experienced under the current ambient temperature, humidity, and solar radiation[17]. In other words, the AT is an adjustment to the dry bulb temperature based on the relative humidity value. Absolute humidity with a dew point of 14°C is chosen as a reference.

[16]. It includes the chilling effect of the wind at lower temperatures.

Two formulas for AT are in use by the Australian Bureau of Meteorology: one includes solar radiation and the other one does not (http://www.bom.gov.au/info/thermal_stress/ , 29 Sep 2021). Please specify q if you want to estimate AT with solar load.

Parameters

- **tdb** (*float*) – dry bulb air temperature,[°C]
- **rh** (*float*) – relative humidity, [%]
- **v** (*float*) – wind speed 10m above ground level, [m/s]
- **q** (*float*) – Net radiation absorbed per unit area of body surface [W/m2]

Other Parameters

round (*boolean, default True*) – if True rounds output value, if False it does not round it

Returns

at (*float*) – apparent temperature, [°C]

Examples

```
>>> from pythermalcomfort.models import at
>>> at(tdb=25, rh=30, v=0.1)
24.1
```

4.1.7 Ankle draft

`pythermalcomfort.models.ankle_draft.ankle_draft`(*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *v_ankle*, *units='SI'*)

Calculates the percentage of thermally dissatisfied people with the ankle draft (0.1 m) above floor level[23]. This equation is only applicable for vr < 0.2 m/s (40 fps).

Parameters

[17] Steadman RG (1984) A universal scale of apparent temperature. J Appl Meteorol Climatol 23:1674–1687

[16] Blazejczyk, K., Epstein, Y., Jendritzky, G., Staiger, H., Tinz, B., 2012. Comparison of UTCI to selected thermal indices. Int. J. Biometeorol. 56, 515–535. DOI: doi.org/10.1007/s00484-011-0453-2

[23] Liu, S., Schiavon, S., Kabanshi, A., Nazaroff, W.W., 2017. Predicted percentage dissatisfied with ankle draft. Indoor Air 27, 852–862. DOI: doi.org/10.1111/ina.12364

- **tdb** (*float*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'

  Note: The air temperature is the average value over two heights: 0.6 m (24 in.) and 1.1 m (43 in.) for seated occupants and 1.1 m (43 in.) and 1.7 m (67 in.) for standing occupants.

- **tr** (*float*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'

- **vr** (*float*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function `pythermalcomfort.utilities.v_relative()`.

- **rh** (*float*) – relative humidity, [%]

- **met** (*float*) – metabolic rate, [met]

- **clo** (*float*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently, the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function `pythermalcomfort.utilities.clo_dynamic()`.

- **v_ankle** (*float*) – air speed at the 0.1 m (4 in.) above the floor, default in [m/s] in [fps] if *units* = 'IP'

- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.

Returns

- **PPD_ad** (*float*) – Predicted Percentage of Dissatisfied occupants with ankle draft, [%]

- **Acceptability** (*bol*) – The ASHRAE 55 2020 standard defines that the value of air speed at the ankle level is acceptable if PPD_ad is lower or equal than 20 %

Examples

```
>>> from pythermalcomfort.models import ankle_draft
>>> results = ankle_draft(25, 25, 0.2, 50, 1.2, 0.5, 0.3, units="SI")
>>> print(results)
{'PPD_ad': 18.5, 'Acceptability': True}
```

4.1.8 Clothing prediction

`pythermalcomfort.models.clo_tout.`**`clo_tout`**`(`*tout*, *units='SI'*`)`

Representative clothing insulation Icl as a function of outdoor air temperature at 06:00 a.m[4].

Parameters

[4] Schiavon, S., & Lee, K. H. (2013). Dynamic predictive clothing insulation models based on outdoor air and indoor operative temperatures. Building and Environment, 59, 250–260. doi.org/10.1016/j.buildenv.2012.08.024

- **tout** (*float, int, or array-like*) – outdoor air temperature at 06:00 a.m., default in [°C] in [°F] if *units* = 'IP'
- **units** (*str, optional*) – select the SI (International System of Units) or the IP (Imperial Units) system. Supported values are 'SI' and 'IP'. Defaults to 'SI'.

Returns
clo (*float, int, or array-like*) – Representative clothing insulation Icl, [clo]

Raises

- `TypeError` – If *tout* is not a float, int, NumPy array, or a list of floats or integers.
- `ValueError` – If an invalid unit is provided or non-numeric elements are found in *tout*.

Notes

The ASHRAE 55 2020 states that it is acceptable to determine the clothing insulation Icl using this equation in mechanically conditioned buildings[Page 8, 1].

Limitations:

- This equation may not be accurate for extreme temperature ranges.

Examples

```
>>> from pythermalcomfort.models import clo_tout
>>> clo_tout(tout=27)  # Can be int or float
0.46
>>> clo_tout(tout=[27, 25])  # List of ints or floats
array([0.46, 0.47])
```

4.1.9 Cooling Effect (CE)

`pythermalcomfort.models.cooling_effect.`**`cooling_effect`**(*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *wme=0*, *units='SI'*)

Returns the value of the Cooling Effect (CE) calculated in compliance with the ASHRAE 55 2020 Standard[Page 8, 1]. The CE of the elevated air speed is the value that, when subtracted equally from both the average air temperature and the mean radiant temperature, yields the same SET under still air as in the first SET calculation under elevated air speed. The cooling effect is calculated only for air speed higher than 0.1 m/s.

Parameters

- **tdb** (*float*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **vr** (*float*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function *`pythermalcomfort.utilities.v_relative()`*.
- **rh** (*float*) – relative humidity, [%]
- **met** (*float*) – metabolic rate, [met]

- **clo** (*float*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function `pythermalcomfort.utilities.clo_dynamic()`.

- **wme** (*float, default 0*) – external work, [met]
- **units** (*{'SI', 'IP'} select the SI (International System of Units) or the IP (Imperial Units) system.*)

Returns

ce (*float*) – Cooling Effect, default in [°C] in [°F] if *units* = 'IP'

Examples

```
>>> from pythermalcomfort.models import cooling_effect
>>> CE = cooling_effect(tdb=25, tr=25, vr=0.3, rh=50, met=1.2, clo=0.5)
>>> print(CE)
1.64

>>> # for users who wants to use the IP system
>>> CE = cooling_effect(tdb=77, tr=77, vr=1.64, rh=50, met=1, clo=0.6, units="IP")
>>> print(CE)
3.74
```

Raises

ValueError – If the cooling effect could not be calculated

4.1.10 Discomfort Index (DI)

`pythermalcomfort.models.discomfort_index.`**`discomfort_index`**`(`*tdb*, *rh*`)`

Calculates the Discomfort Index (DI). The index is essentially an effective temperature based on air temperature and humidity. The discomfort index is usuallly divided in 6 dicomfort categories and it only applies to warm environments:[24]

- class 1 - DI < 21 °C - No discomfort
- class 2 - 21 <= DI < 24 °C - Less than 50% feels discomfort
- class 3 - 24 <= DI < 27 °C - More than 50% feels discomfort
- class 4 - 27 <= DI < 29 °C - Most of the population feels discomfort
- class 5 - 29 <= DI < 32 °C - Everyone feels severe stress
- class 6 - DI >= 32 °C - State of medical emergency

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, [°C]

[24] Polydoros, Anastasios & Cartalis, Constantinos. (2015). Use of Earth Observation based indices for the monitoring of built-up area features and dynamics in support of urban energy studies. Energy and Buildings. 98. 92-99. 10.1016/j.enbuild.2014.09.060.

- **rh** (*float, int, or array-like*) – relative humidity, [%]

Returns

- **di** (*float, int, or array-like*) – Discomfort Index, [°C]
- **discomfort_condition** (*str or array-like*) – Classification of the thermal comfort conditions according to the discomfort index

Examples

```
>>> from pythermalcomfort.models import discomfort_index
>>> discomfort_index(tdb=25, rh=50)
{'di': 22.1, 'discomfort_condition': 'Less than 50% feels discomfort'}
```

4.1.11 Heat Index (HI)

`pythermalcomfort.models.heat_index.`**`heat_index`**`(`*tdb*, *rh*, ***kwargs*`)`

Calculates the Heat Index (HI). It combines air temperature and relative humidity to determine an apparent temperature. The HI equation[12] is derived by multiple regression analysis in temperature and relative humidity from the first version of Steadman's (1979) apparent temperature (AT)[13].

Parameters

- **tdb** (*float*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **rh** (*float*) – relative humidity, [%]

Other Parameters

- **round** (*boolean, default True*) – if True rounds output value, if False it does not round it
- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.

Returns

hi (*float*) – Heat Index, default in [°C] in [°F] if *units* = 'IP'

Examples

```
>>> from pythermalcomfort.models import heat_index
>>> heat_index(tdb=25, rh=50)
25.9
```

[12] Rothfusz LP (1990) The heat index equation. NWS Southern Region Technical Attachment, SR/SSD 90–23, Fort Worth, Texas

[13] Steadman RG (1979) The assessment of sultriness. Part I: A temperature-humidity index based on human physiology and clothing science. J Appl Meteorol 18:861–873

4.1.12 Humidex

`pythermalcomfort.models.humidex.humidex`(*tdb*, *rh*, ***kwargs*)

Calculates the humidex (short for "humidity index"). It has been developed by the Canadian Meteorological service. It was introduced in 1965 and then it was revised by Masterson and Richardson (1979)[14]. It aims to describe how hot, humid weather is felt by the average person. The Humidex differs from the heat index in being related to the dew point rather than relative humidity[15].

Parameters

- **tdb** (*float*) – dry bulb air temperature, [°C]
- **rh** (*float*) – relative humidity, [%]

Other Parameters

round (*boolean, default True*) – if True rounds output value, if False it does not round it

Returns

- **humidex** (*float*) – Heat Index, [°C]
- **discomfort** (*str*) – Degree of Comfort or Discomfort as defined in Havenith and Fiala (2016)[Page 19, 15]

Examples

```
>>> from pythermalcomfort.models import humidex
>>> humidex(tdb=25, rh=50)
{"humidex": 28.2, "discomfort": "Little or no discomfort"}
```

4.1.13 Joint system thermoregulation model (JOS-3)

class `pythermalcomfort.models.jos3.JOS3`(*height=1.72*, *weight=74.43*, *fat=15*, *age=20*, *sex='male'*, *ci=2.59*, *bmr_equation='harris-benedict'*, *bsa_equation='dubois'*, *ex_output=None*)

JOS-3 model simulates human thermal physiology including skin temperature, core temperature, sweating rate, etc. for the whole body and 17 local body parts.

This model was developed at Shin-ichi Tanabe Laboratory, Waseda University and was derived from 65 Multi-Node model (https://doi.org/10.1016/S0378-7788(02)00014-2) and JOS-2 model (https://doi.org/10.1016/j.buildenv.2013.04.013).

To use this model, create an instance of the JOS3 class with optional body parameters such as body height, weight, age, sex, etc.

Environmental conditions such as air temperature, mean radiant temperature, air velocity, etc. can be set using the setter methods. (ex. X.tdb, X.tr X.v) If you want to set the different conditions in each body part, set them as a 17 lengths of list, dictionary, or numpy array format.

List or numpy array format input must be 17 lengths and means the order of "head", "neck", "chest", "back", "pelvis", "left_shoulder", "left_arm", "left_hand", "right_shoulder", "right_arm", "right_hand", "left_thigh", "left_leg", "left_foot", "right_thigh", "right_leg" and "right_foot".

[14] Masterton JM, Richardson FA. Humidex, a method of quantifying human discomfort due to excessive heat and humidity. Downsview, Ontario: CLI 1-79, Environment Canada, Atmospheric Environment Service, 1979

[15] Havenith, G., Fiala, D., 2016. Thermal indices and thermophysiological modeling for heat stress. Compr. Physiol. 6, 255–302. DOI: doi.org/10.1002/cphy.c140051

The model output includes local and mean skin temperature, local core temperature, local and mean skin wettedness, and heat loss from the skin etc. The model output can be accessed using "dict_results()" method and be converted to a csv file using "to_csv" method. Each output parameter also can be accessed using getter methods. (ex. X.t_skin, X.t_skin_mean, X.t_core)

If you use this package, please cite us as follows and mention the version of pythermalcomfort used: Y. Takahashi, A. Nomoto, S. Yoda, R. Hisayama, M. Ogata, Y. Ozeki, S. Tanabe, Thermoregulation Model JOS-3 with New Open Source Code, Energy & Buildings (2020), doi: https://doi.org/10.1016/j.enbuild.2020.110575

Note: To maintain consistency in variable names for pythermalcomfort, some variable names differ from those used in the original paper.

`__init__`(*height=1.72*, *weight=74.43*, *fat=15*, *age=20*, *sex='male'*, *ci=2.59*, *bmr_equation='harris-benedict'*, *bsa_equation='dubois'*, *ex_output=None*)

Initialize a new instance of JOS3 class, which is designed to model and simulate various physiological parameters related to human thermoregulation.

This class uses mathematical models to calculate and predict body temperature, basal metabolic rate, body surface area, and other related parameters.

Parameters

- **height** (*float, optional*) – body height, in [m]. The default is 1.72.
- **weight** (*float, optional*) – body weight, in [kg]. The default is 74.43.
- **fat** (*float, optional*) – fat percentage, in [%]. The default is 15.
- **age** (*int, optional*) – age, in [years]. The default is 20.
- **sex** (*str, optional*) – sex ("male" or "female"). The default is "male".
- **ci** (*float, optional*) – Cardiac index, in [L/min/m2]. The default is 2.6432.
- **bmr_equation** (*str, optional*) – The equation used to calculate basal metabolic rate (BMR). Choose a BMR equation. The default is "harris-benedict" equation created uding Caucasian's data. (DOI: doi.org/10.1073/pnas.4.12.370) If the Ganpule's equation (DOI: doi.org/10.1038/sj.ejcn.1602645) for Japanese people is used, input "japanese".
- **bsa_equation** (*str, optional*) – The equation used to calculate body surface area (bsa). Choose a bsa equation. You can choose "dubois", "fujimoto", "kruazumi", or "takahira". The default is "dubois". The body surface area can be calculated using the function `pythermalcomfort.utilities.body_surface_area()`.
- **ex_output** (*None, list or "all", optional*) – This is used when you want to display results other than the default output parameters (ex.skin temperature); by default, JOS outputs only the most necessary parameters in order to reduce the computational load. If the parameters other than the default output parameters are needed, specify the list of the desired parameter names in string format like ["bf_skin", "bf_core", "t_artery"]. If you want to display all output results, set ex_output is "all".

Variables

- **tdb** (`float, int, or array-like`) – Dry bulb air temperature [°C].
- **tr** (`float, int, or array-like`) – Mean radiant temperature [°C].
- **to** (`float, int, or array-like`) – Operative temperature [°C].
- **v** (`float, int, or array-like`) – Air speed [m/s].
- **rh** (`float, int, or array-like`) – Relative humidity [%].

- `clo` (*float or array-like*) – Clothing insulation [clo]. Note: If you want to input clothing insulation to each body part, it can be input using the dictionaly in "utilities.py" in "jos3_function" folder. `pythermalcomfort.jos3_functions.utilities.local_clo_typical_ensembles()`.
- `par` (*float*) – Physical activity ratio [-]. This equals the ratio of metabolic rate to basal metabolic rate. The par of sitting quietly is 1.2.
- `posture` (*str*) – Body posture [-]. Choose a posture from standing, sitting or lying.
- `body_temp` (*numpy.ndarray (85,)*) – All segment temperatures of JOS-3

`simulate(times, dtime, output):`

Run JOS-3 model for given times.

`dict_results():`

Get results as a dictionary with pandas.DataFrame values.

`to_csv(path=None, folder=None, unit=True, meaning=True):`

Export results as csv format.

Returns

- **cardiac_output** (*cardiac output (the sum of the whole blood flow) [L/h]*)
- **cycle_time** (*the counts of executing one cycle calculation [-]*)
- **dt** (*time step [sec]*)
- **pythermalcomfort_version** (*version of pythermalcomfort [-]*)
- **q_res** (*heat loss by respiration [W]*)
- **q_skin2env** (*total heat loss from the skin (each body part) [W]*)
- **q_thermogenesis_total** (*total thermogenesis of the whole body [W]*)
- **simulation_time** (*simulation times [sec]*)
- **t_core** (*core temperature (each body part) [°C]*)
- **t_skin** (*skin temperature (each body part) [°C]*)
- **t_skin_mean** (*mean skin temperature [°C]*)
- **w** (*skin wettedness (each body part) [-]*)
- **w_mean** (*mean skin wettedness [-]*)
- **weight_loss_by_evap_and_res** (*weight loss by the evaporation and respiration of the whole body [g/sec]*)
- **OPTIONAL PARAMETERS** (*the paramters listed below are returned if ex_output = "all"*)
- **age** (*age [years]*)
- **bf_ava_foot** (*AVA blood flow rate of one foot [L/h]*)
- **bf_ava_hand** (*AVA blood flow rate of one hand [L/h]*)
- **bf_core** (*core blood flow rate (each body part) [L/h]*)
- **bf_fat** (*fat blood flow rate (each body part) [L/h]*)
- **bf_muscle** (*muscle blood flow rate (each body part) [L/h]*)

- **bf_skin** (*skin blood flow rate (each body part) [L/h]*)
- **bsa** (*body surface area (each body part) [m2]*)
- **clo** (*clothing insulation (each body part) [clo]*)
- **e_max** (*maximum evaporative heat loss from the skin (each body part) [W]*)
- **e_skin** (*evaporative heat loss from the skin (each body part) [W]*)
- **e_sweat** (*evaporative heat loss from the skin by only sweating (each body part) [W]*)
- **fat** (*body fat rate [%]*)
- **height** (*body height [m]*)
- **name** (*name of the model [-]*)
- **par** (*physical activity ratio [-]*)
- **q_bmr_core** (*core thermogenesis by basal metabolism (each body part) [W]*)
- **q_bmr_fat** (*fat thermogenesis by basal metabolism (each body part) [W]*)
- **q_bmr_muscle** (*muscle thermogenesis by basal metabolism (each body part) [W]*)
- **q_bmr_skin** (*skin thermogenesis by basal metabolism (each body part) [W]*)
- **q_nst** (*core thermogenesis by non-shivering (each body part) [W]*)
- **q_res_latent** (*latent heat loss by respiration (each body part) [W]*)
- **q_res_sensible** (*sensible heat loss by respiration (each body part) [W]*)
- **q_shiv** (*core or muscle thermogenesis by shivering (each body part) [W]*)
- **q_skin2env_latent** (*latent heat loss from the skin (each body part) [W]*)
- **q_skin2env_sensible** (*sensible heat loss from the skin (each body part) [W]*)
- **q_thermogenesis_core** (*core total thermogenesis (each body part) [W]*)
- **q_thermogenesis_fat** (*fat total thermogenesis (each body part) [W]*)
- **q_thermogenesis_muscle** (*muscle total thermogenesis (each body part) [W]*)
- **q_thermogenesis_skin** (*skin total thermogenesis (each body part) [W]*)
- **q_work** (*core or muscle thermogenesis by work (each body part) [W]*)
- **r_et** (*total clothing evaporative heat resistance (each body part) [(m2*kPa)/W]*)
- **r_t** (*total clothing heat resistance (each body part) [(m2*K)/W]*)
- **rh** (*relative humidity (each body part) [%]*)
- **sex** (*sex [-]*)
- **t_artery** (*arterial temperature (each body part) [°C]*)
- **t_cb** (*central blood temperature [°C]*)
- **t_core_set** (*core set point temperature (each body part) [°C]*)
- **t_fat** (*fat temperature (each body part) [°C]*)
- **t_muscle** (*muscle temperature (each body part) [°C]*)
- **t_skin_set** (*skin set point temperature (each body part) [°C]*)
- **t_superficial_vein** (*superficial vein temperature (each body part) [°C]*)

- **t_vein** (*vein temperature (each body part) [°C]*)
- **tdb** (*dry bulb air temperature (each body part) [°C]*)
- **to** (*operative temperature (each body part) [°C]*)
- **tr** (*mean radiant temperature (each body part) [°C]*)
- **v** (*air velocity (each body part) [m/s]*)
- **weight** (*body weight [kg]*)

Examples

Build a model and set a body built Create an instance of the JOS3 class with optional body parameters such as body height, weight, age, sex, etc.

```
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> import os
>>> from pythermalcomfort.models import JOS3
>>> from pythermalcomfort.jos3_functions.utilities import local_clo_typical_
↪ensembles
>>>
>>> directory_name = "jos3_output_example"
>>> current_directory = os.getcwd()
>>> jos3_example_directory = os.path.join(current_directory, directory_name)
>>> if not os.path.exists(jos3_example_directory):
>>>     os.makedirs(jos3_example_directory)
>>>
>>> model = JOS3(
>>>     height=1.7,
>>>     weight=60,
>>>     fat=20,
>>>     age=30,
>>>     sex="male",
>>>     bmr_equation="japanese",
>>>     bsa_equation="fujimoto",
>>>     ex_output="all",
>>> )
>>> # Set environmental conditions such as air temperature, mean radiant␣
↪temperature using the setter methods.
>>> # Set the first condition
>>> # Environmental parameters can be input as int, float, list, dict, numpy␣
↪array format.
>>> model.tdb = 28  # Air temperature [°C]
>>> model.tr = 30  # Mean radiant temperature [°C]
>>> model.rh = 40  # Relative humidity [%]
>>> model.v = np.array( # Air velocity [m/s]
>>>     [
>>>         0.2,  # head
>>>         0.4,  # neck
>>>         0.4,  # chest
>>>         0.1,  # back
```

(continues on next page)

(continued from previous page)

```
>>>         0.1,  # pelvis
>>>         0.4,  # left shoulder
>>>         0.4,  # left arm
>>>         0.4,  # left hand
>>>         0.4,  # right shoulder
>>>         0.4,  # right arm
>>>         0.4,  # right hand
>>>         0.1,  # left thigh
>>>         0.1,  # left leg
>>>         0.1,  # left foot
>>>         0.1,  # right thigh
>>>         0.1,  # right leg
>>>         0.1,  # right foot
>>>     ]
>>> )
>>> model.clo = local_clo_typical_ensembles["briefs, socks, undershirt, work␣
↪jacket, work pants, safety shoes"]["local_body_part"]
>>> # par should be input as int, float.
>>> model.par = 1.2  # Physical activity ratio [-], assuming a sitting position
>>> # posture should be input as int (0, 1, or 2) or str ("standing", "sitting"␣
↪or "lying").
>>> # (0="standing", 1="sitting" or 2="lying")
>>> model.posture = "sitting"  # Posture [-], assuming a sitting position
>>>
>>> # Run JOS-3 model
>>> model.simulate(
>>>     times=30,  # Number of loops of a simulation
>>>     dtime=60,  # Time delta [sec]. The default is 60.
>>> )  # Exposure time = 30 [loops] * 60 [sec] = 30 [min]
>>> # Set the next condition (You only need to change the parameters that you␣
↪want to change)
>>> model.to = 20  # Change operative temperature
>>> model.v = { # Air velocity [m/s], assuming to use a desk fan
>>>     'head' : 0.2,
>>>     'neck' : 0.4,
>>>     'chest' : 0.4,
>>>     'back': 0.1,
>>>     'pelvis' : 0.1,
>>>     'left_shoulder' : 0.4,
>>>     'left_arm' : 0.4,
>>>     'left_hand' : 0.4,
>>>     'right_shoulder' : 0.4,
>>>     'right_arm' : 0.4,
>>>     'right_hand' : 0.4,
>>>     'left_thigh' : 0.1,
>>>     'left_leg' : 0.1,
>>>     'left_foot' : 0.1,
>>>     'right_thigh' : 0.1,
>>>     'right_leg' : 0.1,
>>>     'right_foot' : 0.1
>>>     }
>>> # Run JOS-3 model
```

(continues on next page)

(continued from previous page)

```
>>> model.simulate(
>>>     times=60,  # Number of loops of a simulation
>>>     dtime=60,  # Time delta [sec]. The default is 60.
>>> )  # Additional exposure time = 60 [loops] * 60 [sec] = 60 [min]
>>> # Set the next condition (You only need to change the parameters that you␣
↪want to change)
>>> model.tdb = 30  # Change air temperature [°C]
>>> model.tr = 35  # Change mean radiant temperature [°C]
>>> # Run JOS-3 model
>>> model.simulate(
>>>     times=30,  # Number of loops of a simulation
>>>     dtime=60,  # Time delta [sec]. The default is 60.
>>> )  # Additional exposure time = 30 [loops] * 60 [sec] = 30 [min]
>>> # Show the results
>>> df = pd.DataFrame(model.dict_results())  # Make pandas.DataFrame
>>> df[["t_skin_mean", "t_skin_head", "t_skin_chest", "t_skin_left_hand"]].
↪plot()  # Plot time series of local skin temperature.
>>> plt.legend(["Mean", "Head", "Chest", "Left hand"])  # Reset the legends
>>> plt.ylabel("Skin temperature [°C]")  # Set y-label as 'Skin temperature [°C]'
>>> plt.xlabel("Time [min]")  # Set x-label as 'Time [min]'
>>> plt.savefig(os.path.join(jos3_example_directory, "jos3_example2_skin_
↪temperatures.png"))  # Save plot at the current directory
>>> plt.show()  # Show the plot
>>> # Exporting the results as csv
>>> model.to_csv(os.path.join(jos3_example_directory, "jos3_example2 (all␣
↪output).csv"))
```

property bmr

float Basal metabolic rate [W/m2].

Type
bmr

property body_names

list JOS3 body names

Type
body_names

property body_temp

numpy.ndarray (85,) All segment temperatures of JOS-3

Type
body_temp

property bsa

numpy.ndarray (17,) Body surface areas by local body segments [m2].

Type
bsa

property clo

numpy.ndarray (17,) Clothing insulation [clo].

Type
clo

`dict_results()`

Get results as a dictionary with pandas.DataFrame values.

Returns
dict – A dictionary of the results, with keys as column names and values as pandas.DataFrame objects.

`property par`

float Physical activity ratio [-].This equals the ratio of metabolic rate to basal metabolic rate. par of sitting quietly is 1.2.

Type
par

`property posture`

str Current JOS3 posture.

Type
posture

`property r_et`

numpy.ndarray (17,) w (Evaporative) heat resistances between the skin and ambience areas by local body segments [(m2*kPa)/W].

Type
r_et

`property r_t`

numpy.ndarray (17,) Dry heat resistances between the skin and ambience areas by local body segments [(m2*K)/W].

Type
r_t

`property results`

dict.

Type
Results of the model

`property rh`

numpy.ndarray (17,) Relative humidity [%].

Type
rh

`simulate`(*times*, *dtime=60*, *output=True*)

Run JOS-3 model.

Parameters

- **times** (*int*) – Number of loops of a simulation.
- **dtime** (*int or float, optional*) – Time delta in seconds. The default is 60.
- **output** (*bool, optional*) – If you don't want to record parameters, set False. The default is True.

Returns
None.

property t_artery

numpy.ndarray (17,) Arterial temperatures by the local body segments [°C].

Type
t_artery

property t_cb

numpy.ndarray (1,) Temperature at central blood pool [°C].

Type
t_cb

property t_core

numpy.ndarray (17,) Skin temperatures by the local body segments [°C].

Type
t_core

property t_fat

numpy.ndarray (2,) fat temperatures of head and pelvis [°C].

Type
t_fat

property t_muscle

numpy.ndarray (2,) Muscle temperatures of head and pelvis [°C].

Type
t_muscle

property t_skin

numpy.ndarray (17,) Skin temperatures by the local body segments [°C].

Type
t_skin

property t_skin_mean

float Mean skin temperature of the whole body [°C].

Type
t_skin_mean

property t_superficial_vein

numpy.ndarray (12,) Superficial vein temperatures by the local body segments [°C].

Type
t_superficial_vein

property t_vein

numpy.ndarray (17,) Vein temperatures by the local body segments [°C].

Type
t_vein

property tdb

Dry-bulb air temperature. The setter accepts int, float, dict, list, ndarray. The inputs are used to create a 17-element array. dict should be passed with BODY_NAMES as keys.

Returns
ndarray – A NumPy array of shape (17,).

`property to`

numpy.ndarray (17,) Operative temperature [°C].

Type
to

`to_csv`(*path=None*, *folder=None*, *unit=True*, *meaning=True*)

Export results as csv format.

Parameters

- **path** (*str, optional*) – Output path. If you don't use the default file name, set a name. The default is None.
- **folder** (*str, optional*) – Output folder. If you use the default file name with the current time, set a only folder path. The default is None.
- **unit** (*bool, optional*) – Write units in csv file. The default is True.
- **meaning** (*bool, optional*) – Write meanings of the parameters in csv file. The default is True.

Returns
None

Examples

```
>>> from pythermalcomfort.models import JOS3
>>> model = JOS3()
>>> model.simulate(60)
>>> model.to_csv()
```

`property tr`

numpy.ndarray (17,) Mean radiant temperature [°C].

Type
tr

`property v`

numpy.ndarray (17,) Air velocity [m/s].

Type
v

`property version`

float The current version of pythermalcomfort.

Type
version

`property w`

numpy.ndarray (17,) Skin wettedness on local body segments [-].

Type
w

`property w_mean`

float Mean skin wettedness of the whole body [-].

Type
w_mean

4.1.14 Normal Effective Temperature (NET)

`pythermalcomfort.models.net.`**`net`**`(`*tdb*, *rh*, *v*, ***kwargs*`)`

Calculates the Normal Effective Temperature (NET). Missenard (1933) devised a formula for calculating effective temperature. The index establishes a link between the same condition of the organism's thermoregulatory capability (warm and cold perception) and the surrounding environment's temperature and humidity. The index is calculated as a function of three meteorological factors: air temperature, relative humidity of air, and wind speed. This index allows to calculate the effective temperature felt by a person. Missenard original equation was then used to calculate the Normal Effective Temperature (NET), by considering normal atmospheric pressure and a normal human body temperature (37°C). The NET is still in use in Germany, where medical check-ups for subjects working in the heat are decided on by prevailing levels of ET, depending on metabolic rates. The NET is also constantly monitored by the Hong Kong Observatory[Page 14, 16]. In central Europe the following thresholds are in use: <1°C = very cold; 1–9 = cold; 9–17 = cool; 17–21 = fresh; 21–23 = comfortable; 23–27 = warm; >27°C = hot[Page 14, 16].

Parameters

- **tdb** (*float,*) – dry bulb air temperature, [°C]
- **rh** (*float*) – relative humidity, [%]
- **v** (*float*) – wind speed [m/s] at 1.2 m above the ground

Other Parameters
round (*boolean, default True*) – if True rounds output value, if False it does not round it

Returns
net (*float*) – Normal Effective Temperature, [°C]

Examples

```
>>> from pythermalcomfort.models import net
>>> net(tdb=37, rh=100, v=0.1)
37
```

4.1.15 Predicted Heat Strain (PHS) Index

`pythermalcomfort.models.phs.`**`phs`**`(`*tdb*, *tr*, *v*, *rh*, *met*, *clo*, *posture*, *wme=0*, ***kwargs*`)`

Calculates the Predicted Heat Strain (PHS) index based in compliace with the ISO 7933:2004 Standard[8]. The ISO 7933 provides a method for the analytical evaluation and interpretation of the thermal stress experienced by a subject in a hot environment. It describes a method for predicting the sweat rate and the internal core temperature that the human body will develop in response to the working conditions.

The PHS model can be used to predict the: heat by respiratory convection, heat flow by respiratory evaporation, steady state mean skin temperature, instantaneous value of skin temperature, heat accumulation associated with the metabolic rate, maximum evaporative heat flow at the skin surface, predicted sweat rate, predicted evaporative heat flow, and rectal temperature.

Parameters

[8] ISO, 2004. ISO 7933 - Ergonomics of the thermal environment — Analytical determination and interpretation of heat stress using calculation of the predicted heat strain.

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C]
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C]
- **v** (*float, int, or array-like*) – air speed, default in [m/s]
- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **met** (*float, int, or array-like*) – metabolic rate, [W/(m2)]
- **clo** (*float, int, or array-like*) – clothing insulation, [clo]
- **posture** (*int*) – a numeric value presenting posture of person [sitting=1, standing=2, crouching=3]
- **wme** (*float, int, or array-like*) – external work, [W/(m2)] default 0

Other Parameters

- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns values even if input values are outside the applicability limits of the model.

  The 7933 limits are 15 < tdb [°C] < 50, 0 < tr [°C] < 60, 0 < vr [m/s] < 3, 100 < met [met] < 450, and 0.1 < clo [clo] < 1.
- **i_mst** (*float, default 0.38*) – static moisture permeability index, [dimensionless]
- **a_p** (*float, default 0.54*) – fraction of the body surface covered by the reflective clothing, [dimensionless]
- **drink** (*int, default 1*) – 1 if workers can drink freely, 0 otherwise
- **weight** (*float, default 75*) – body weight, [kg]
- **height** (*float, default 1.8*) – height, [m]
- **walk_sp** (*float, default 0*) – walking speed, [m/s]
- **theta** (*float, default 0*) – angle between walking direction and wind direction [degrees]
- **acclimatized** (*int, default 100*) – 100 if acclimatized subject, 0 otherwise
- **duration** (*int, default 480*) – duration of the work sequence, [minutes]
- **f_r** (*float, default 0.97*) – emissivity of the reflective clothing, [dimensionless] Some reference values `pythermalcomfort.utilities.f_r_garments()`.
- **t_sk** (*float, default 34.1*) – mean skin temperature when worker starts working, [°C]
- **t_cr** (*float, default 36.8*) – mean core temperature when worker starts working, [°C]
- **t_re** (*float, default False*) – mean rectal temperature when worker starts working, [°C]
- **t_cr_eq** (*float, default False*) – mean core temperature as a function of met when worker starts working, [°C]
- **sweat_rate** (*float, default 0*)

Returns

- **t_re** (*float*) – rectal temperature, [°C]
- **t_sk** (*float*) – skin temperature, [°C]
- **t_cr** (*float*) – core temperature, [°C]
- **t_cr_eq** (*float*) – core temperature as a function of the metabolic rate, [°C]

- **t_sk_t_cr_wg** (*float*) – fraction of the body mass at the skin temperature
- **d_lim_loss_50** (*float*) – maximum allowable exposure time for water loss, mean subject, [minutes]
- **d_lim_loss_95** (*float*) – maximum allowable exposure time for water loss, 95% of the working population, [minutes]
- **d_lim_t_re** (*float*) – maximum allowable exposure time for heat storage, [minutes]
- **water_loss_watt** (*float*) – maximum water loss in watts, [W]
- **water_loss** (*float*) – maximum water loss, [g]

Examples

```
>>> from pythermalcomfort.models import phs
>>> results = phs(tdb=40, tr=40, rh=33.85, v=0.3, met=150, clo=0.5, posture=2,␣
↪wme=0)
>>> print(results)
{'t_re': 37.5, 'd_lim_loss_50': 440, 'd_lim_loss_95': 298, 'd_lim_t_re': 480,
'water_loss': 6166.0}
```

4.1.16 Physiological Equivalent Temperature (PET)

`pythermalcomfort.models.pet_steady.`**`pet_steady`**(*tdb*, *tr*, *v*, *rh*, *met*, *clo*, *p_atm=1013.25*, *position=1*, *age=23*, *sex=1*, *weight=75*, *height=1.8*, *wme=0*)

The steady physiological equivalent temperature (PET) is calculated using the Munich Energy-balance Model for Individuals (MEMI), which simulates the human body's thermal circumstances in a medically realistic manner. PET is defined as the air temperature at which, in a typical indoor setting the heat budget of the human body is balanced with the same core and skin temperature as under the complex outdoor conditions to be assessed[20]. The following assumptions are made for the indoor reference climate: tdb = tr, v = 0.1 m/s, water vapour pressure = 12 hPa, clo = 0.9 clo, and met = 1.37 met + basic metabolism. PET allows a layperson to compare the total effects of complex thermal circumstances outside with his or her own personal experience indoors in this way. This function solves the heat balances without accounting for heat storage in the human body.

The PET was originally proposed by Hoppe[Page 31, 20]. In 2018, Walther and Goestchel[21] proposed a correction of the original model, purging the errors in the PET calculation routine, and implementing a state-of-the-art vapour diffusion model. Walther and Goestchel (2018) model is therefore used to calculate the PET.

Parameters

- **tdb** (*float*) – dry bulb air temperature, [°C]
- **tr** (*float*) – mean radiant temperature, [°C]
- **v** (*float*) – air speed, [m/s]
- **rh** (*float*) – relative humidity, [%]
- **met** (*float*) – metabolic rate, [met]
- **clo** (*float*) – clothing insulation, [clo]

[20] Höppe P. The physiological equivalent temperature - a universal index for the biometeorological assessment of the thermal environment. Int J Biometeorol. 1999 Oct;43(2):71-5. doi: 10.1007/s004840050118. PMID: 10552310.

[21] Walther, E. and Goestchel, Q., 2018. The PET comfort index: Questioning the model. Building and Environment, 137, pp.1-10. DOI: doi.org/10.1016/j.buildenv.2018.03.054

- **p_atm** (*float*) – atmospheric pressure, default value 1013.25 [hPa]
- **position** (*int*) – position of the individual (1=sitting, 2=standing, 3=standing, forced convection)
- **age** (*int, default 23*) – age in years
- **sex** (*int, default 1*) – male (1) or female (2).
- **weight** (*float, default 75*) – body mass, [kg]
- **height** (*float, default 1.8*) – height, [m]
- **wme** (*float, default 0*) – external work, [W/(m2)] default 0

Returns

PET – Steady-state PET under the given ambient conditions

Examples

```
>>> from pythermalcomfort.models import pet_steady
>>> pet_steady(tdb=20, tr=20, rh=50, v=0.15, met=1.37, clo=0.5)
18.85
```

4.1.17 Predicted Mean Vote (PMV) and Predicted Percentage of Dissatisfied (PPD)

pythermalcomfort.models.pmv_ppd.**pmv_ppd**(*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *wme=0*, *standard='ISO'*, *units='SI'*, *limit_inputs=True*, *airspeed_control=True*)

Returns Predicted Mean Vote (PMV) and Predicted Percentage of Dissatisfied (PPD) calculated in accordance to main thermal comfort Standards. The PMV is an index that predicts the mean value of the thermal sensation votes (self-reported perceptions) of a large group of people on a sensation scale expressed from –3 to +3 corresponding to the categories: cold, cool, slightly cool, neutral, slightly warm, warm, and hot.[Page 8, 1]

While the PMV equation is the same for both the ISO and ASHRAE standards, in the ASHRAE 55 PMV equation, the SET is used to calculate the cooling effect first, this is then subtracted from both the air and mean radiant temperatures, and the differences are used as input to the PMV model, while the airspeed is set to 0.1m/s. Please read more in the Note below.

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **vr** (*float, int, or array-like*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function `pythermalcomfort.utilities.v_relative()`.
- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **met** (*float, int, or array-like*) – metabolic rate, [met]
- **clo** (*float, int, or array-like*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently, the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function *`pythermalcomfort.utilities.clo_dynamic()`*.

- **wme** (*float, int, or array-like*) – external work, [met] default 0
- **standard** (*str, optional*) – select comfort standard used for calculation. Supported values are 'ASHRAE' and 'ISO'. Defaults to 'ISO'.
  - If "ISO", then the ISO Equation is used
  - If "ASHRAE", then the ASHRAE Equation is used

  Note: While the PMV equation is the same for both the ISO and ASHRAE standards, the ASHRAE Standard Use of the PMV model is limited to air speeds below 0.10 m/s (20 fpm). When air speeds exceed 0.10 m/s (20 fpm), the comfort zone boundaries are adjusted based on the SET model. This change was indroduced by the Addendum C to Standard 55-2020
- **units** (*str, optional*) – select the SI (International System of Units) or the IP (Imperial Units) system. Supported values are 'SI' and 'IP'. Defaults to 'SI'.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

  The ASHRAE 55 2020 limits are 10 < tdb [°C] < 40, 10 < tr [°C] < 40, 0 < vr [m/s] < 2, 1 < met [met] < 4, and 0 < clo [clo] < 1.5. The ISO 7730 2005 limits are 10 < tdb [°C] < 30, 10 < tr [°C] < 40, 0 < vr [m/s] < 1, 0.8 < met [met] < 4, 0 < clo [clo] < 2, and -2 < PMV < 2.
- **airspeed_control** (*boolean default True*) – This only applies if standard = "ASHRAE". By default it is assumed that the occupant has control over the airspeed. In this case the ASHRAE 55 Standard does not impose any airspeed limits. On the other hand, if the occupant has no control over the airspeed the ASHRAE 55 imposes an upper limit for v which varies as a function of the operative temperature, for more information please consult the Standard.

Returns

- **pmv** (*float, int, or array-like*) – Predicted Mean Vote
- **ppd** (*float, int, or array-like*) – Predicted Percentage of Dissatisfied occupants, [%]

Notes

You can use this function to calculate the PMV and PPD in accordance with either the ASHRAE 55 2020 Standard[Page 8, 1] or the ISO 7730 Standard[Page 11, 2].

Examples

```
>>> from pythermalcomfort.models import pmv_ppd
>>> from pythermalcomfort.utilities import v_relative, clo_dynamic
>>> tdb = 25
>>> tr = 25
>>> rh = 50
>>> v = 0.1
>>> met = 1.4
>>> clo = 0.5
```

(continues on next page)

(continued from previous page)

```
>>> # calculate relative air speed
>>> v_r = v_relative(v=v, met=met)
>>> # calculate dynamic clothing
>>> clo_d = clo_dynamic(clo=clo, met=met)
>>> results = pmv_ppd(tdb=tdb, tr=tr, vr=v_r, rh=rh, met=met, clo=clo_d)
>>> print(results)
{'pmv': 0.06, 'ppd': 5.1}
>>> print(results["pmv"])
-0.06
>>> # you can also pass an array-like of inputs
>>> results = pmv_ppd(tdb=[22, 25], tr=tr, vr=v_r, rh=rh, met=met, clo=clo_d)
>>> print(results)
{'pmv': array([-0.47,  0.06]), 'ppd': array([9.6, 5.1])}
```

Raises

- **StopIteration** – Raised if the number of iterations exceeds the threshold
- **ValueError** – The 'standard' function input parameter can only be 'ISO' or 'ASHRAE'

4.1.18 Predicted Mean Vote (PMV)

pythermalcomfort.models.pmv.**pmv**(*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *wme=0*, *standard='ISO'*, *units='SI'*, *limit_inputs=True*, *airspeed_control=True*)

Returns Predicted Mean Vote (PMV) calculated in accordance to main thermal comfort Standards. The PMV is an index that predicts the mean value of the thermal sensation votes (self-reported perceptions) of a large group of people on a sensation scale expressed from –3 to +3 corresponding to the categories: cold, cool, slightly cool, neutral, slightly warm, warm, and hot.[Page 8, 1]

While the PMV equation is the same for both the ISO and ASHRAE standards, in the ASHRAE 55 PMV equation, the SET is used to calculate the cooling effect first, this is then subtracted from both the air and mean radiant temperatures, and the differences are used as input to the PMV model, while the airspeed is set to 0.1m/s. Please read more in the Note below.

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **vr** (*float, int, or array-like*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function `pythermalcomfort.utilities.v_relative()`.

- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **met** (*float, int, or array-like*) – metabolic rate, [met]
- **clo** (*float, int, or array-like*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently, the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function `pythermalcomfort.utilities.clo_dynamic()`.

- **wme** (*float, int, or array-like*) – external work, [met] default 0
- **standard** (*str, optional*) – select comfort standard used for calculation. Supported values are 'ASHRAE' and 'ISO'. Defaults to 'ISO'.
  - If "ISO", then the ISO Equation is used
  - If "ASHRAE", then the ASHRAE Equation is used

  Note: While the PMV equation is the same for both the ISO and ASHRAE standards, the ASHRAE Standard Use of the PMV model is limited to air speeds below 0.10 m/s (20 fpm). When air speeds exceed 0.10 m/s (20 fpm), the comfort zone boundaries are adjusted based on the SET model. This change was indroduced by the Addendum C to Standard 55-2020
- **units** (*str, optional*) – select the SI (International System of Units) or the IP (Imperial Units) system. Supported values are 'SI' and 'IP'. Defaults to 'SI'.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

  The ASHRAE 55 2020 limits are 10 < tdb [°C] < 40, 10 < tr [°C] < 40, 0 < vr [m/s] < 2, 1 < met [met] < 4, and 0 < clo [clo] < 1.5. The ISO 7730 2005 limits are 10 < tdb [°C] < 30, 10 < tr [°C] < 40, 0 < vr [m/s] < 1, 0.8 < met [met] < 4, 0 < clo [clo] < 2, and -2 < PMV < 2.
- **airspeed_control** (*boolean default True*) – This only applies if standard = "ASHRAE". By default it is assumed that the occupant has control over the airspeed. In this case the ASHRAE 55 Standard does not impose any airspeed limits. On the other hand, if the occupant has no control over the airspeed the ASHRAE 55 imposes an upper limit for v which varies as a function of the operative temperature, for more information please consult the Standard.

Returns

pmv (*float, int, or array-like*) – Predicted Mean Vote

Notes

You can use this function to calculate the PMV[Page 8, 1][Page 11, 2].

Examples

```
>>> from pythermalcomfort.models import pmv
>>> from pythermalcomfort.utilities import v_relative, clo_dynamic
>>> t_db = 25
>>> t_r = 25
>>> relative_humidity = 50
>>> v = 0.1
>>> met_rate = 1.4
>>> clo_insulation = 0.5
>>> # calculate relative air speed
>>> v_r = v_relative(v=v, met=met_rate)
>>> # calculate dynamic clothing
>>> clo_d = clo_dynamic(clo=clo_insulation, met=met_rate)
```

(continues on next page)

(continued from previous page)

```
>>> results = pmv(tdb=t_db, tr=t_r, vr=v_r, rh=relative_humidity, met=met_rate,␣
↪clo=clo_d)
>>> print(results)
0.06
>>> # you can also pass an array-like of inputs
>>> results = pmv(tdb=[22, 25], tr=tr, vr=v_r, rh=rh, met=met, clo=clo_d)
>>> print(results)
array([-0.47,  0.06])
```

4.1.19 Solar gain on people

`pythermalcomfort.models.solar_gain.solar_gain`(*sol_altitude*, *sharp*, *sol_radiation_dir*, *sol_transmittance*, *f_svv*, *f_bes*, *asw=0.7*, *posture='seated'*, *floor_reflectance=0.6*)

Calculates the solar gain to the human body using the Effective Radiant Field (ERF)[Page 8, 1]. The ERF is a measure of the net energy flux to or from the human body. ERF is expressed in W over human body surface area [w/m2]. In addition, it calculates the delta mean radiant temperature. Which is the amount by which the mean radiant temperature of the space should be increased if no solar radiation is present.

Parameters

- **sol_altitude** (*float*) – Solar altitude, degrees from horizontal [deg]. Ranges between 0 and 90.
- **sharp** (*float*) – Solar horizontal angle relative to the front of the person (SHARP) [deg]. Ranges between 0 and 180 and is symmetrical on either side. Zero (0) degrees represents direct-beam radiation from the front, 90 degrees represents direct-beam radiation from the side, and 180 degrees rep- resent direct-beam radiation from the back. SHARP is the angle between the sun and the person only. Orientation relative to compass or to room is not included in SHARP.
- **posture** (*str*) – Default 'seated' list of available options 'standing', 'supine' or 'seated'
- **sol_radiation_dir** (*float*) – Direct-beam solar radiation, [W/m2]. Ranges between 200 and 1000. See Table C2-3 of ASHRAE 55 2020[Page 8, 1].
- **sol_transmittance** (*float*) – Total solar transmittance, ranges from 0 to 1. The total solar transmittance of window systems, including glazing unit, blinds, and other façade treatments, shall be determined using one of the following methods: i) Provided by manufacturer or from the National Fenestration Rating Council approved Lawrence Berkeley National Lab International Glazing Database. ii) Glazing unit plus venetian blinds or other complex or unique shades shall be calculated using National Fenestration Rating Council approved software or Lawrence Berkeley National Lab Complex Glazing Database.
- **f_svv** (*float*) – Fraction of sky-vault view fraction exposed to body, ranges from 0 to 1. It can be calculated using the function `pythermalcomfort.utilities.f_svv()`.
- **f_bes** (*float*) – Fraction of the possible body surface exposed to sun, ranges from 0 to 1. See Table C2-2 and equation C-7 ASHRAE 55 2020[Page 8, 1].
- **asw** (*float*) – The average short-wave absorptivity of the occupant. It will range widely, depending on the color of the occupant's skin as well as the color and amount of clothing covering the body. A value of 0.7 shall be used unless more specific information about the clothing or skin color of the occupants is available. Note: Short-wave absorptivity typically

ranges from 0.57 to 0.84, depending on skin and clothing color. More information is available in Blum (1945).

- **floor_reflectance** (*float*) – Floor refectance. It is assumed to be constant and equal to 0.6.

Notes

More information on the calculation procedure can be found in Appendix C of[Page 8, 1].

Returns

- **erf** (*float*) – Solar gain to the human body using the Effective Radiant Field [W/m2]
- **delta_mrt** (*float*) – Delta mean radiant temperature. The amount by which the mean radiant temperature of the space should be increased if no solar radiation is present.

Examples

```
>>> from pythermalcomfort.models import solar_gain
>>> results = solar_gain(sol_altitude=0, sharp=120,
sol_radiation_dir=800, sol_transmittance=0.5, f_svv=0.5, f_bes=0.5,
asw=0.7, posture='seated')
>>> print(results)
{'erf': 42.9, 'delta_mrt': 10.3}
```

4.1.20 Standard Effective Temperature (SET)

`pythermalcomfort.models.set_tmp.set_tmp`(*tdb*, *tr*, *v*, *rh*, *met*, *clo*, *wme=0*, *body_surface_area=1.8258*, *p_atm=101325*, *body_position='standing'*, *units='SI'*, *limit_inputs=True*, ***kwargs*)

Calculates the Standard Effective Temperature (SET). The SET is the temperature of a hypothetical isothermal environment at 50% (rh), <0.1 m/s (20 fpm) average air speed (v), and tr = tdb, in which the total heat loss from the skin of an imaginary occupant wearing clothing, standardized for the activity concerned is the same as that from a person in the actual environment with actual clothing and activity level.[10]

Parameters

- **tdb** (*float or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **v** (*float or array-like*) – air speed, default in [m/s] in [fps] if *units* = 'IP'
- **rh** (*float or array-like*) – relative humidity, [%]
- **met** (*float or array-like*) – metabolic rate, [met]
- **clo** (*float or array-like*) – clothing insulation, [clo]
- **wme** (*float or array-like*) – external work, [met] default 0
- **body_surface_area** (*float*) – body surface area, default value 1.8258 [m2] in [ft2] if *units* = 'IP'

[10] Gagge, A.P., Fobelets, A.P., and Berglund, L.G., 1986. A standard predictive Index of human reponse to thermal enviroment. Am. Soc. Heating, Refrig. Air-Conditioning Eng. 709–731.

The body surface area can be calculated using the function `pythermalcomfort.utilities.body_surface_area()`.

- **p_atm** (*float*) – atmospheric pressure, default value 101325 [Pa] in [atm] if *units* = 'IP'
- **body_position** (*str default="standing" or array-like*) – select either "sitting" or "standing"
- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the following limits the function returns nan. If False returns values regardless of the input values. The limits are 10 < tdb [°C] < 40, 10 < tr [°C] < 40, 0 < v [m/s] < 2, 1 < met [met] < 4, and 0 < clo [clo] < 1.5.

Other Parameters

round (*boolean, deafult True*) – if True rounds output value, if False it does not round it

Returns

SET (*float or array-like*) – Standard effective temperature, [°C]

Notes

You can use this function to calculate the SET temperature in accordance with the ASHRAE 55 2020 Standard[Page 8, 1].

Examples

```
>>> from pythermalcomfort.models import set_tmp
>>> set_tmp(tdb=25, tr=25, v=0.1, rh=50, met=1.2, clo=.5)
24.3
>>> set_tmp(tdb=[25, 25], tr=25, v=0.1, rh=50, met=1.2, clo=.5)
array([24.3, 24.3])

>>> # for users who wants to use the IP system
>>> set_tmp(tdb=77, tr=77, v=0.328, rh=50, met=1.2, clo=.5, units='IP')
75.8
```

4.1.21 Two-node model

`pythermalcomfort.models.two_nodes.`**`two_nodes`**(*tdb*, *tr*, *v*, *rh*, *met*, *clo*, *wme=0*, *body_surface_area=1.8258*, *p_atmospheric=101325*, *body_position='standing'*, *max_skin_blood_flow=90*, ***kwargs*)

Two-node model of human temperature regulation Gagge et al. (1986).[Page 37, 10] This model it can be used to calculate a variety of indices, including:

- Gagge's version of Fanger's Predicted Mean Vote (PMV). This function uses the Fanger's PMV equations but it replaces the heat loss and gain terms with those calculated by the two node model developed by Gagge et al. (1986)[Page 37, 10].
- PMV SET and the predicted thermal sensation based on SET[Page 37, 10]. This function is similar in all aspects to the `pythermalcomfort.models.pmv_gagge()` however, it uses the `pythermalcomfort.models.set()` equation to calculate the dry heat loss by convection.

- Thermal discomfort (DISC) as the relative thermoregulatory strain necessary to restore a state of comfort and thermal equilibrium by sweating[Page 37, 10]. DISC is described numerically as: comfortable and pleasant (0), slightly uncomfortable but acceptable (1), uncomfortable and unpleasant (2), very uncomfortable (3), limited tolerance (4), and intolerable (S). The range of each category is $\pm$ 0.5 numerically. In the cold, the classical negative category descriptions used for Fanger's PMV apply[Page 37, 10].
- Heat gains and losses via convection, radiation and conduction.
- The Standard Effective Temperature (SET)
- The New Effective Temperature (ET)
- The Predicted Thermal Sensation (TSENS)
- The Predicted Percent Dissatisfied Due to Draft (PD)
- Predicted Percent Satisfied With the Level of Air Movement" (PS)

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **v** (*float or array-like*) – air speed, default in [m/s] in [fps] if *units* = 'IP'
- **rh** (*float or array-like*) – relative humidity, [%]
- **met** (*float or array-like*) – metabolic rate, [met]
- **clo** (*float or array-like*) – clothing insulation, [clo]
- **wme** (*float or array-like*) – external work, [met] default 0
- **body_surface_area** (*float*) – body surface area, default value 1.8258 [m2] in [ft2] if *units* = 'IP'

  The body surface area can be calculated using the function `pythermalcomfort.utilities.body_surface_area()`.
- **p_atmospheric** (*float*) – atmospheric pressure, default value 101325 [Pa] in [atm] if *units* = 'IP'
- **body_position** (*str default="standing" or array-like*) – select either "sitting" or "standing"
- **max_skin_blood_flow** (*float*) – maximum blood flow from the core to the skin, [kg/h/m2] default 80

Other Parameters

- **round** (*boolean, default True*) – if True rounds output values, if False it does not round them
- **max_sweating** (*float*) – Maximum rate at which regulatory sweat is generated, [kg/h/m2]
- **w_max** (*float*) – Maximum skin wettedness (w) adimensional. Ranges from 0 and 1.

Returns

- **e_skin** (*float or array-like*) – Total rate of evaporative heat loss from skin, [W/m2]. Equal to e_rsw + e_diff
- **e_rsw** (*float or array-like*) – Rate of evaporative heat loss from sweat evaporation, [W/m2]
- **e_diff** (*float or array-like*) – Rate of evaporative heat loss from moisture diffused through the skin, [W/m2]
- **e_max** (*float or array-like*) – Maximum rate of evaporative heat loss from skin, [W/m2]

- **q_sensible** (*float or array-like*) – Sensible heat loss from skin, [W/m2]
- **q_skin** (*float or array-like*) – Total rate of heat loss from skin, [W/m2]. Equal to q_sensible + e_skin
- **q_res** (*float or array-like*) – Total rate of heat loss through respiration, [W/m2]
- **t_core** (*float or array-like*) – Core temperature, [°C]
- **t_skin** (*float or array-like*) – Skin temperature, [°C]
- **m_bl** (*float or array-like*) – Skin blood flow, [kg/h/m2]
- **m_rsw** (*float or array-like*) – Rate at which regulatory sweat is generated, [kg/h/m2]
- **w** (*float or array-like*) – Skin wettedness, adimensional. Ranges from 0 and 1.
- **w_max** (*float or array-like*) – Skin wettedness (w) practical upper limit, adimensional. Ranges from 0 and 1.
- **set** (*float or array-like*) – Standard Effective Temperature (SET)
- **et** (*float or array-like*) – New Effective Temperature (ET)
- **pmv_gagge** (*float or array-like*) – PMV Gagge
- **pmv_set** (*float or array-like*) – PMV SET
- **pd** (*float or array-like*) – Predicted Percent Dissatisfied Due to Draft”
- **ps** (*float or array-like*) – Predicted Percent Satisfied With the Level of Air Movement
- **disc** (*float or array-like*) – Thermal discomfort
- **t_sens** (*float or array-like*) – Predicted Thermal Sensation

Examples

```
>>> from pythermalcomfort.models import two_nodes
>>> print(two_nodes(tdb=25, tr=25, v=0.3, rh=50, met=1.2, clo=0.5))
{'e_skin': 15.8, 'e_rsw': 6.5, 'e_diff': 9.3, ... }
>>> print(two_nodes(tdb=[25, 25], tr=25, v=0.3, rh=50, met=1.2, clo=0.5))
{'e_skin': array([15.8, 15.8]), 'e_rsw': array([6.5, 6.5]), ... }
```

4.1.22 Universal Thermal Climate Index (UTCI)

`pythermalcomfort.models.utci.`**`utci`**(*tdb*, *tr*, *v*, *rh*, *units='SI'*, *return_stress_category=False*, *limit_inputs=True*)

Determines the Universal Thermal Climate Index (UTCI). The UTCI is the equivalent temperature for the environment derived from a reference environment. It is defined as the air temperature of the reference environment which produces the same strain index value in comparison with the reference individual’s response to the real environment. It is regarded as one of the most comprehensive indices for calculating heat stress in outdoor spaces. The parameters that are taken into account for calculating UTCI involve dry bulb temperature, mean radiation temperature, the pressure of water vapor or relative humidity, and wind speed (at the elevation of 10 m above the ground).[7]

Parameters

[7] Zare, S., Hasheminejad, N., Shirvan, H.E., Hemmatjo, R., Sarebanzadeh, K., Ahmadi, S., 2018. Comparing Universal Thermal Climate Index (UTCI) with selected thermal indices/environmental parameters during 12 months of the year. Weather Clim. Extrem. 19, 49–57. https://doi.org/10.1016/j.wace.2018.01.004

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float, int, or array-like*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **v** (*float, int, or array-like*) – wind speed 10m above ground level, default in [m/s] in [fps] if *units* = 'IP'
- **rh** (*float, int, or array-like*) – relative humidity, [%]
- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.
- **return_stress_category** (*boolean default False*) – if True returns the UTCI categorized in terms of thermal stress.
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outsude the standard applicability limits the function returns nan. If False returns UTCI values even if input values are outside the applicability limits of the model. The valid input ranges are -50 < tdb [°C] < 50, tdb - 70 < tr [°C] < tdb + 30, and for 0.5 < v [m/s] < 17.0.

Returns

- **utci** (*float, int, or array-like*) – Universal Thermal Climate Index, [°C] or in [°F]
- **stress_category** (*str or array-like*) – UTCI categorized in terms of thermal stress[9].

Notes

You can use this function to calculate the Universal Thermal Climate Index (*UTCI*) The applicability wind speed value must be between 0.5 and 17 m/s.

Examples

```
>>> from pythermalcomfort.models import utci
>>> utci(tdb=25, tr=25, v=1.0, rh=50)
24.6

>>> # for users who wants to use the IP system
>>> utci(tdb=77, tr=77, v=3.28, rh=50, units='ip')
76.4

>>> # for users who wants to get stress category
>>> utci(tdb=25, tr=25, v=1.0, rh=50, return_stress_category=True)
{"utci": 24.6, "stress_category": "no thermal stress"}
```

Raises

`ValueError` – Raised if the input are outside the Standard's applicability limits

[9] Błażejczyk, K., Jendritzky, G., Bröde, P., Fiala, D., Havenith, G., Epstein, Y., Psikuta, A. and Kampmann, B., 2013. An introduction to the universal thermal climate index (UTCI). Geographia Polonica, 86(1), pp.5-10.

4.1.23 Use Fans During Heatwaves

pythermalcomfort.models.use_fans_heatwaves.**use_fans_heatwaves**(*tdb*, *tr*, *v*, *rh*, *met*, *clo*, *wme=0*, *body_surface_area=1.8258*, *p_atm=101325*, *body_position='standing'*, *units='SI'*, *max_skin_blood_flow=80*, ***kwargs*)

It helps you to estimate if the conditions you have selected would cause heat strain. This occurs when either the following variables reaches its maximum value:

- m_rsw Rate at which regulatory sweat is generated, [mL/h/m2]
- w : Skin wettedness, adimensional. Ranges from 0 and 1.
- m_bl : Skin blood flow [kg/h/m2]

Parameters

- **tdb** (*float*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'
- **tr** (*float*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **v** (*float*) – air speed, default in [m/s] in [fps] if *units* = 'IP'
- **rh** (*float*) – relative humidity, [%]
- **met** (*float*) – metabolic rate, [met]
- **clo** (*float*) – clothing insulation, [clo]
- **wme** (*float*) – external work, [met] default 0
- **body_surface_area** (*float*) – body surface area, default value 1.8258 [m2] in [ft2] if *units* = 'IP'

  The body surface area can be calculated using the function `pythermalcomfort.utilities.body_surface_area()`.
- **p_atm** (*float*) – atmospheric pressure, default value 101325 [Pa] in [atm] if *units* = 'IP'
- **body_position** (*str default="standing"*) – select either "sitting" or "standing"
- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.
- **max_skin_blood_flow** (*float, [kg/h/m2] default 80*) – maximum blood flow from the core to the skin

Other Parameters

- **max_sweating** (*float, [mL/h/m2] default 500*) – max sweating
- **round** (*boolean, default True*) – if True rounds output value, if False it does not round it
- **limit_inputs** (*boolean default True*) – By default, if the inputs are outside the standard applicability limits the function returns nan. If False returns pmv and ppd values even if input values are outside the applicability limits of the model.

  The applicability limits are 20 < tdb [°C] < 50, 20 < tr [°C] < 50, 0.1 < v [m/s] < 4.5, 0.7 < met [met] < 2, and 0 < clo [clo] < 1.

Returns

- **e_skin** (*float*) – Total rate of evaporative heat loss from skin, [W/m2]. Equal to e_rsw + e_diff
- **e_rsw** (*float*) – Rate of evaporative heat loss from sweat evaporation, [W/m2]
- **e_diff** (*float*) – Rate of evaporative heat loss from moisture diffused through the skin, [W/m2]
- **e_max** (*float*) – Maximum rate of evaporative heat loss from skin, [W/m2]
- **q_sensible** (*float*) – Sensible heat loss from skin, [W/m2]
- **q_skin** (*float*) – Total rate of heat loss from skin, [W/m2]. Equal to q_sensible + e_skin
- **q_res** (*float*) – Total rate of heat loss through respiration, [W/m2]
- **t_core** (*float*) – Core temperature, [°C]
- **t_skin** (*float*) – Skin temperature, [°C]
- **m_bl** (*float*) – Skin blood flow, [kg/h/m2]
- **m_rsw** (*float*) – Rate at which regulatory sweat is generated, [mL/h/m2]
- **w** (*float*) – Skin wettedness, adimensional. Ranges from 0 and 1.
- **w_max** (*float*) – Skin wettedness (w) practical upper limit, adimensional. Ranges from 0 and 1.
- **heat_strain** (*bool*) – True if the model predict that the person may be experiencing heat strain
- **heat_strain_blood_flow** (*bool*) – True if heat strain is caused by skin blood flow (m_bl) reaching its maximum value
- **heat_strain_w** (*bool*) – True if heat strain is caused by skin wettedness (w) reaching its maximum value
- **heat_strain_sweating** (*bool*) – True if heat strain is caused by regulatory sweating (m_rsw) reaching its maximum value

4.1.24 Vertical air temperature gradient

`pythermalcomfort.models.vertical_tmp_grad_ppd.`**`vertical_tmp_grad_ppd`**`(`*tdb*, *tr*, *vr*, *rh*, *met*, *clo*, *vertical_tmp_grad*, *units='SI'*`)`

Calculates the percentage of thermally dissatisfied people with a vertical temperature gradient between feet and head[Page 8, 1]. This equation is only applicable for vr < 0.2 m/s (40 fps).

Parameters

- **tdb** (*float*) – dry bulb air temperature, default in [°C] in [°F] if *units* = 'IP'

  Note: The air temperature is the average value over two heights: 0.6 m (24 in.) and 1.1 m (43 in.) for seated occupants and 1.1 m (43 in.) and 1.7 m (67 in.) for standing occupants.
- **tr** (*float*) – mean radiant temperature, default in [°C] in [°F] if *units* = 'IP'
- **vr** (*float*) – relative air speed, default in [m/s] in [fps] if *units* = 'IP'

  Note: vr is the relative air speed caused by body movement and not the air speed measured by the air speed sensor. The relative air speed is the sum of the average air speed measured by the sensor plus the activity-generated air speed (Vag). Where Vag is the activity-generated air speed caused by motion of individual body parts. vr can be calculated using the function `pythermalcomfort.utilities.v_relative()`.

- **rh** (*float*) – relative humidity, [%]
- **met** (*float*) – metabolic rate, [met]
- **clo** (*float*) – clothing insulation, [clo]

  Note: The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met) The dynamic clothing insulation, clo, can be calculated using the function `pythermalcomfort.utilities.clo_dynamic()`.
- **vertical_tmp_grad** (*float*) – vertical temperature gradient between the feet and the head, default in [°C/m] in [°F/ft] if *units* = 'IP'
- **units** (*{'SI', 'IP'}*) – select the SI (International System of Units) or the IP (Imperial Units) system.

Returns

- **PPD_vg** (*float*) – Predicted Percentage of Dissatisfied occupants with vertical temperature gradient, [%]
- **Acceptability** (*bol*) – The ASHRAE 55 2020 standard defines that the value of air speed at the ankle level is acceptable if PPD_ad is lower or equal than 5 %

Examples

```
>>> from pythermalcomfort.models import vertical_tmp_grad_ppd
>>> results = vertical_tmp_grad_ppd(25, 25, 0.1, 50, 1.2, 0.5, 7)
>>> print(results)
{'PPD_vg': 12.6, 'Acceptability': False}
```

4.1.25 Wet Bulb Globe Temperature Index (WBGT)

`pythermalcomfort.models.wbgt.`**`wbgt`**(*twb*, *tg*, *tdb=None*, *with_solar_load=False*, ***kwargs*)

Calculates the Wet Bulb Globe Temperature (WBGT) index calculated in compliance with the ISO 7243[11]. The WBGT is a heat stress index that measures the thermal environment to which a person is exposed. In most situations, this index is simple to calculate. It should be used as a screening tool to determine whether heat stress is present. The PHS model allows a more accurate estimation of stress. PHS can be calculated using the function `pythermalcomfort.models.phs()`.

The WBGT determines the impact of heat on a person throughout the course of a working day (up to 8 h). It does not apply to very brief heat exposures. It pertains to the evaluation of male and female people who are fit for work in both indoor and outdoor occupational environments, as well as other sorts of surroundings[11].

The WBGT is defined as a function of only twb and tg if the person is not exposed to direct radiant heat from the sun. When a person is exposed to direct radiant heat, tdb must also be specified.

Parameters

- **twb** (*float,*) – natural (no forced air flow) wet bulb temperature, [°C]
- **tg** (*float*) – globe temperature, [°C]

[11] ISO, 2017. ISO 7243 - Ergonomics of the thermal environment — Assessment of heat stress using the WBGT (wet bulb globe temperature) index.

- **tdb** (*float*) – dry bulb air temperature, [°C]. This value is needed as input if the person is exposed to direct solar radiation
- **with_solar_load** (*bool*) – True if the globe sensor is exposed to direct solar radiation

Other Parameters

round (*boolean, default True*) – if True rounds output value, if False it does not round it

Returns

wbgt (*float*) – Wet Bulb Globe Temperature Index, [°C]

Examples

```
>>> from pythermalcomfort.models import wbgt
>>> wbgt(twb=25, tg=32)
27.1

>>> # if the persion is exposed to direct solar radiation
>>> wbgt(twb=25, tg=32, tdb=20, with_solar_load=True)
25.9
```

4.1.26 Wind chill index

`pythermalcomfort.models.wc.wc(tdb, v, **kwargs)`

Calculates the Wind Chill Index (WCI) in accordance with the ASHRAE 2017 Handbook Fundamentals - Chapter 9[18].

The wind chill index (WCI) is an empirical index based on cooling measurements taken on a cylindrical flask partially filled with water in Antarctica (Siple and Passel 1945). For a surface temperature of 33°C, the index describes the rate of heat loss from the cylinder via radiation and convection as a function of ambient temperature and wind velocity.

This formulation has been met with some valid criticism. WCI is unlikely to be an accurate measure of heat loss from exposed flesh, which differs from plastic in terms of curvature, roughness, and radiation exchange qualities, and is always below 33°C in a cold environment. Furthermore, the equation's values peak at 90 km/h and then decline as velocity increases. Nonetheless, this score reliably represents the combined effects of temperature and wind on subjective discomfort for velocities below 80 km/h[Page 45, 18].

Parameters

- **tdb** (*float*) – dry bulb air temperature,[°C]
- **v** (*float*) – wind speed 10m above ground level, [m/s]

Other Parameters

round (*boolean, default True*) – if True rounds output value, if False it does not round it

Returns

wci (*float*) – wind chill index, [W/m2)]

[18] ASHRAE, 2017. 2017 ASHRAE Handbook Fundamentals. Atlanta.

Examples

```
>>> from pythermalcomfort.models import wc
>>> wc(tdb=-5, v=5.5)
{"wci": 1255.2}
```

4.2 Psychrometrics functions

class pythermalcomfort.psychrometrics.**PsychrometricValues**(*p_sat: Union[float, int, numpy.ndarray, List[float], List[int]], p_vap: Union[float, int, numpy.ndarray, List[float], List[int]], hr: Union[float, int, numpy.ndarray, List[float], List[int]], t_wb: Union[float, int, numpy.ndarray, List[float], List[int]], t_dp: Union[float, int, numpy.ndarray, List[float], List[int]], h: Union[float, int, numpy.ndarray, List[float], List[int]]*)

pythermalcomfort.psychrometrics.**enthalpy**(*tdb*, *hr*)

Calculates air enthalpy

Parameters

- **tdb** (*float, int, or array-like*) – air temperature, [°C]
- **hr** (*float, int, or array-like*) – humidity ratio, [kg water/kg dry air]

Returns

enthalpy (*float, int, or array-like*) – enthalpy [J/kg dry air]

pythermalcomfort.psychrometrics.**p_sat**(*tdb*)

Calculates vapour pressure of water at different temperatures

Parameters

tdb (*float, int, or array-like*) – air temperature, [°C]

Returns

p_sat (*float, int, or array-like*) – saturation vapor pressure, [Pa]

pythermalcomfort.psychrometrics.**p_sat_torr**(*tdb*)

Estimates the saturation vapor pressure in [torr]

Parameters

tdb (*float, int, or array-like*) – dry bulb air temperature, [C]

Returns

p_sat (*float*) – saturation vapor pressure [torr]

pythermalcomfort.psychrometrics.**psy_ta_rh**(*tdb*, *rh*, *p_atm=101325*)

Calculates psychrometric values of air based on dry bulb air temperature and relative humidity. For more accurate results we recommend the use of the Python package psychrolib.

Parameters

- **tdb** (*float, int, or array-like*) – air temperature, [°C]
- **rh** (*float, int, or array-like*) – relative humidity, [%]

- **p_atm** (*float, int, or array-like*) – atmospheric pressure, [Pa]

Returns

- **p_vap** (*float, int, or array-like*) – partial pressure of water vapor in moist air, [Pa]
- **hr** (*float, int, or array-like*) – humidity ratio, [kg water/kg dry air]
- **t_wb** (*float, int, or array-like*) – wet bulb temperature, [°C]
- **t_dp** (*float, int, or array-like*) – dew point temperature, [°C]
- **h** (*float, int, or array-like*) – enthalpy [J/kg dry air]

`pythermalcomfort.psychrometrics.`**`t_dp`**`(`*tdb*, *rh*`)`

Calculates the dew point temperature.

Parameters

- **tdb** (*float, int, or array-like*) – dry bulb air temperature, [°C]
- **rh** (*float, int, or array-like*) – relative humidity, [%]

Returns

t_dp (*float, int, or array-like*) – dew point temperature, [°C]

`pythermalcomfort.psychrometrics.`**`t_mrt`**`(`*tg*, *tdb*, *v*, *d=0.15*, *emissivity=0.95*, *standard='Mixed Convection'*`)`

Converts globe temperature reading into mean radiant temperature in accordance with either the Mixed Convection developed by Teitelbaum E. et al. (2022) or the ISO 7726:1998 Standard[5].

Parameters

- **tg** (*float, int, or array-like*) – globe temperature, [°C]
- **tdb** (*float, int, or array-like*) – air temperature, [°C]
- **v** (*float, int, or array-like*) – air speed, [m/s]
- **d** (*float, int, or array-like*) – diameter of the globe, [m] default 0.15 m
- **emissivity** (*float, int, or array-like*) – emissivity of the globe temperature sensor, default 0.95
- **standard** (*str, optional*) – Supported values are 'Mixed Convection' and 'ISO'. Defaults to 'Mixed Convection'. either choose between the Mixed Convection and ISO formulations. The Mixed Convection formulation has been proposed by Teitelbaum E. et al. (2022) to better determine the free and forced convection coefficient used in the calculation of the mean radiant temperature. They also showed that mean radiant temperature measured with ping-pong ball-sized globe thermometers is not reliable due to a stochastic convective bias[22]. The Mixed Convection model has only been validated for globe sensors with a diameter between 0.04 and 0.15 m.

Returns

tr (*float, int, or array-like*) – mean radiant temperature, [°C]

`pythermalcomfort.psychrometrics.`**`t_o`**`(`*tdb*, *tr*, *v*, *standard='ISO'*`)`

Calculates operative temperature in accordance with ISO 7726:1998[Page 47, 5]

Parameters

- **tdb** (*float, int, or array-like*) – air temperature, [°C]

[5] ISO. (1998). ISO 7726 - Ergonomics of the thermal environment instruments for measuring physical quantities.

[22] Teitelbaum, E., Alsaad, H., Aviv, D., Kim, A., Voelker, C., Meggers, F., & Pantelic, J. (2022). Addressing a systematic error correcting for free and mixed convection when measuring mean radiant temperature with globe thermometers. Scientific Reports, 12(1), 1–18. DOI: doi.org/10.1038/s41598-022-10172-5

- **tr** (*float, int, or array-like*) – mean radiant temperature, [°C]
- **v** (*float, int, or array-like*) – air speed, [m/s]
- **standard** (*str (default="ISO")*) – either choose between ISO and ASHRAE

Returns
to (*float*) – operative temperature, [°C]

`pythermalcomfort.psychrometrics.`**`t_wb`**`(`*tdb*, *rh*`)`

Calculates the wet-bulb temperature using the Stull equation[6]

Parameters

- **tdb** (*float, int, or array-like*) – air temperature, [°C]
- **rh** (*float, int, or array-like*) – relative humidity, [%]

Returns
tdb (*float, int, or array-like*) – wet-bulb temperature, [°C]

4.3 Utilities functions

4.3.1 Body Surface Area

`pythermalcomfort.utilities.`**`body_surface_area`**`(`*weight*, *height*, *formula='dubois'*`)`

Returns the body surface area in square meters.

Parameters

- **weight** (*float*) – body weight, [kg]
- **height** (*float*) – height, [m]
- **formula** (*str, optional,*) – formula used to calculate the body surface area. default="dubois" Choose a name from "dubois", "takahira", "fujimoto", or "kurazumi".

Returns
body_surface_area (*float*) – body surface area, [m2]

4.3.2 Relative air speed

`pythermalcomfort.utilities.`**`v_relative`**`(`*v*, *met*`)`

Estimates the relative air speed which combines the average air speed of the space plus the relative air speed caused by the body movement. Vag is assumed to be 0 for metabolic rates equal and lower than 1 met and otherwise equal to Vag = 0.3 (M – 1) (m/s)

Parameters

- **v** (*float, int, or array-like*) – air speed measured by the sensor, [m/s]
- **met** (*float*) – metabolic rate, [met]

Returns
vr (*float, int, or array-like*) – relative air speed, [m/s]

[6] Stull, R., 2011. Wet-Bulb Temperature from Relative Humidity and Air Temperature. J. Appl. Meteorol. Climatol. 50, 2267–2269. doi.org/10.1175/JAMC-D-11-0143.1

4.3.3 Dynamic clothing

`pythermalcomfort.utilities.`**`clo_dynamic`**(*clo*, *met*, *standard='ASHRAE'*)

Estimates the dynamic clothing insulation of a moving occupant. The activity as well as the air speed modify the insulation characteristics of the clothing and the adjacent air layer. Consequently, the ISO 7730 states that the clothing insulation shall be corrected[Page 11, 2]. The ASHRAE 55 Standard corrects for the effect of the body movement for met equal or higher than 1.2 met using the equation clo = Icl × (0.6 + 0.4/met)

Parameters

- **clo** (*float, int, or array-like*) – clothing insulation, [clo]
- **met** (*float, int, or array-like*) – metabolic rate, [met]
- **standard** (*str (default="ASHRAE")*) –
  - If "ASHRAE", uses Equation provided in Section 5.2.2.2 of ASHRAE 55 2020

Returns

clo (*float, int, or array-like*) – dynamic clothing insulation, [clo]

4.3.4 Running mean outdoor temperature

`pythermalcomfort.utilities.`**`running_mean_outdoor_temperature`**(*temp_array*, *alpha=0.8*, *units='SI'*)

Estimates the running mean temperature also known as prevailing mean outdoor temperature.

Parameters

- **temp_array** (*list*) – array containing the mean daily temperature in descending order (i.e. from newest/yesterday to oldest) $[t_{day-1}, t_{day-2}, ..., t_{day-n}]$. Where t_{day-1} is yesterday's daily mean temperature. The EN 16798-1 2019[Page 9, 3] states that n should be equal to 7
- **alpha** (*float*) – constant between 0 and 1. The EN 16798-1 2019[Page 9, 3] recommends a value of 0.8, while the ASHRAE 55 2020 recommends to choose values between 0.9 and 0.6, corresponding to a slow- and fast- response running mean, respectively. Adaptive comfort theory suggests that a slow-response running mean (alpha = 0.9) could be more appropriate for climates in which synoptic-scale (day-to- day) temperature dynamics are relatively minor, such as the humid tropics.
- **units** (*str default="SI"*) – select the SI (International System of Units) or the IP (Imperial Units) system.

Returns

t_rm (*float*) – running mean outdoor temperature

4.3.5 Units converter

`pythermalcomfort.utilities.`**`units_converter`**(*from_units='ip'*, ***kwargs*)

Converts IP values to SI units.

Parameters

- **from_units** (*str*) – specify system to convert from
- ****kwargs** (*[t, v]*)

Returns

converted values in SI units

4.3.6 Sky-vault view fraction

`pythermalcomfort.utilities.`**`f_svv`**(*w*, *h*, *d*)

Calculates the sky-vault view fraction.

Parameters

- **w** (*float*) – width of the window, [m]
- **h** (*float*) – height of the window, [m]
- **d** (*float*) – distance between the occupant and the window, [m]

Returns

f_svv (*float*) – sky-vault view fraction ranges between 0 and 1

4.4 Reference values clo and met

4.4.1 Met typical tasks, [met]

`pythermalcomfort.utilities.`**`met_typical_tasks = {'Basketball': 6.3, 'Calisthenics': 3.5, 'Cooking': 1.8, 'Dancing': 3.4, 'Driving a car': 1.5, 'Driving, heavy vehicle': 3.2, 'Filing, seated': 1.2, 'Filing, standing': 1.4, 'Flying aircraft, combat': 2.4, 'Flying aircraft, routine': 1.2, 'Handling 100lb (45 kg) bags': 4.0, 'Heavy machine work': 4.0, 'House cleaning': 2.7, 'Lifting/packing': 2.1, 'Light machine work': 2.2, 'Pick and shovel work': 4.4, 'Reading, seated': 1.0, 'Reclining': 0.8, 'Seated, heavy limb movement': 2.2, 'Seated, quiet': 1.0, 'Sleeping': 0.7, 'Standing, relaxed': 1.2, 'Table sawing': 1.8, 'Tennis': 3.8, 'Typing': 1.1, 'Walking 2mph (3.2kmh)': 2.0, 'Walking 3mph (4.8kmh)': 2.6, 'Walking 4mph (6.4kmh)': 3.8, 'Walking about': 1.7, 'Wrestling': 7.8, 'Writing': 1.0}`**

Met values of typical tasks.

Example

```
>>> from pythermalcomfort.utilities import met_typical_tasks
>>> print(met_typical_tasks['Filing, standing'])
1.4
```

4.4.2 Clothing insulation of typical ensembles, [clo]

`pythermalcomfort.utilities.`**`clo_typical_ensembles = {'Jacket, Trousers, long-sleeve shirt': 0.96, 'Knee-length skirt, long-sleeve shirt, full slip': 0.67, 'Knee-length skirt, short-sleeve shirt, sandals, underwear': 0.54, 'Sweat pants, long-sleeve sweatshirt': 0.74, 'Trousers, long-sleeve shirt': 0.61, 'Trousers, short-sleeve shirt, socks, shoes, underwear': 0.57, 'Typical summer indoor clothing': 0.5, 'Typical winter indoor clothing': 1.0, 'Walking shorts, short-sleeve shirt': 0.36}`**

Total clothing insulation of typical ensembles.

Example

```
>>> from pythermalcomfort.utilities import clo_typical_ensembles
>>> print(clo_typical_ensembles['Typical summer indoor clothing'])
0.5
```

4.4.3 Insulation of individual garments, [clo]

```
pythermalcomfort.utilities.clo_individual_garments = {'Ankle socks': 0.02, 'Boots': 0.1, 'Bra': 0.01, 'Calf length socks': 0.03, 'Coveralls': 0.49, 'Double-breasted coat (thick)': 0.48, 'Double-breasted coat (thin)': 0.42, 'Executive chair': 0.15, 'Full slip': 0.16, 'Half slip': 0.14, 'Knee socks (thick)': 0.06, 'Long sleeve shirt (thick)': 0.36, 'Long sleeve shirt (thin)': 0.25, 'Long underwear bottoms': 0.15, 'Long underwear top': 0.2, 'Long-sleeve dress shirt': 0.25, 'Long-sleeve flannel shirt': 0.34, 'Long-sleeve long gown': 0.46, 'Long-sleeve long wrap robe (thick)': 0.69, 'Long-sleeve pajamas (thick)': 0.57, 'Long-sleeve shirt dress (thick)': 0.47, 'Long-sleeve shirt dress (thin)': 0.33, 'Long-sleeve short wrap robe (thick)': 0.48, 'Long-sleeve sweat shirt': 0.34, "Men's underwear": 0.04, 'Metal chair': 0.0, 'Overalls': 0.3, 'Panty hose': 0.02, 'Shoes or sandals': 0.02, 'Short shorts': 0.06, 'Short-sleeve dress shirt': 0.19, 'Short-sleeve hospital gown': 0.31, 'Short-sleeve knit shirt': 0.17, 'Short-sleeve pajamas': 0.42, 'Short-sleeve shirt dress': 0.29, 'Short-sleeve short robe (thin)': 0.34, 'Single-breasted coat (thick)': 0.44, 'Single-breasted coat (thin)': 0.36, 'Sleeveless long gown (thin)': 0.2, 'Sleeveless scoop-neck blouse': 0.12, 'Sleeveless short gown (thin)': 0.18, 'Sleeveless vest (thick)': 0.17, 'Sleeveless vest (thin)': 0.1, 'Sleeveless, scoop-neck shirt (thick)': 0.27, 'Sleeveless, scoop-neck shirt (thin)': 0.23, 'Slippers': 0.03, 'Standard office chair': 0.1, 'Sweatpants': 0.28, 'T-shirt': 0.08, 'Thick skirt': 0.23, 'Thick trousers': 0.24, 'Thin skirt': 0.14, 'Thin trousers': 0.15, 'Walking shorts': 0.08, "Women's underwear": 0.03, 'Wooden stool': 0.01}
```

Clo values of individual clothing elements. To calculate the total clothing insulation you need to add these values together.

Example

```
>>> from pythermalcomfort.utilities import clo_individual_garments
>>> print(clo_individual_garments['T-shirt'])
0.08

>>> # calculate total clothing insulation
>>> i_cl = clo_individual_garments['T-shirt'] + clo_individual_garments["Men's underwear
↪"] +
>>>        clo_individual_garments['Thin trousers'] + clo_individual_garments['Shoes or␣
↪sandals']
>>> print(i_cl)
0.29
```

References

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

pythermalcomfort could always use more documentation, whether as part of the official pythermalcomfort docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/CenterForTheBuiltEnvironment/pythermalcomfort/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *pythermalcomfort* for local development:

1. Fork pythermalcomfort (look for the “Fork” button).
2. Clone your fork locally. Fetch and pull all the updates from the master branch before you do anything:

```
git clone git@github.com:CenterForTheBuiltEnvironment/pythermalcomfort.git
```

3. Create a branch for local development. The naming rule for new branch are, as follows:
   - If this update is for a new feature Feature/feature_name_here
   - If this update is for bug fix Fix/bug_name_here
   - If this update is for documentation Documentation/doc_name_here

You can create a branch locally using the following command. Make sure you only push updates to this new branch only:

```
git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you’re done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request after you have done all your modifications and tested your work. The pull request should include a detailed description of your work:
   - What this pull request is about
   - Have you tested your work
   - Will this work affect other component in the product

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you’re developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there’s new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

To add a function

1. Add the function to the python file *pythermalcomfort/models/* and document it.
2. Add any related functions that are used by your function either in *pythermalcomfort/utilities.py* or *src/pythermalcomfort/psychrometrics.py*. See existing code as example.
3. Test your function by writing a test in *tests/test_XXXX.py*. Test it by running tox -e pyXX where XX is the Python version you want to use, e.g. 37
4. Add *autofunction* to *doc.reference.pythermalcomfort.py*.

AUTHORS

- Federico Tartarini
- Stefano Schiavon
- Tyler Hoyt
- Chris Mackey

Derivative work

pythermalcomfort is a derivative work of the following software projects:

- CBE Comfort Tool for indoor thermal comfort calculations. Available under GPL.
- ladybug-comfort. Available under GPL.
- UTCI Fortran Code for outdoor thermal comfort calculations. Available under MIT.

CHAPTER SEVEN

CONTACT US

If you need help, would like to ask a question about the tool, give us a feedback, or suggest a new feature you can now use GitHub discussions.

7.1 Bug reports

You can report bugs using GitHub issues. When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

CHAPTER

EIGHT

CHANGELOG

8.1 2.9.1 (2024-01-19)

- Fixed error calculation of mass sweating in PET mode, the unit was incorrect

8.2 2.9.0 (2024-01-15)

Warning: pythermalcomfort 2.9.0 is no longer compatible with Python 3.8

- The PHS model accepts arrays as inputs

8.3 2.8.11 (2023-10-26)

- wrote more test and improved code

8.4 2.8.11 (2023-10-26)

- fixed issues with the documentation and sorted the models in alphabetical order

8.5 2.8.7 (2023-10-23)

- Adaptive ASHRAE now returns a dataclass

8.6 2.8.6 (2023-10-09)

- re-structured and linted the code

8.7 2.8.4 (2023-09-20)

- calculation of cooling effect in pmv (standard='ashrae') triggered only when v>0.1 m/s

8.8 2.8.3 (2023-09-14)

- general improvements in the JOS3 model

8.9 2.8.2 (2023-09-04)

- general improvements in the JOS3 model
- fixed error when e_max == 0

8.10 2.8.1 (2023-07-05)

- pythermalcomfort needs Python version > 3.8
- fixed issue in Cooling Effect calculation

8.11 2.8.0 (2023-07-03)

- allowing the cooling effect to range from 0 to 40
- fixed PHS documentation
- improved JOS3 documentation

8.12 2.7.0 (2023-02-16)

- changed coefficient of vasodilation in set_tmp() to 120 to match ASHRAE 55 2020 code
- slightly modified value in validation tables

8.13 2.6.0 (2023-01-17)

- max sweating rate can be passed to two node model
- max skin wettedness can be passed to two node model
- rounding w to two decimals
- use_fans_heatwave function accepts arrays
- fixed typos unit documentation

8.14 2.5.4 (2022-10-12)

- PHS model accepts all required inputs to be run on a minute by minute basis
- fix error check compliance PHS model

8.15 2.5.0 (2022-06-13)

- Added the adaptive thermal heat balance (ATHB) model

8.16 2.4.0 (2022-06-10)

- Added e_pmv model - Adjusted Predicted Mean Votes with Expectancy Factor
- Added a_pmv model - Adaptive Predicted Mean Vote

8.17 2.3.0 (2022-06-01)

- Added discomfort index

8.18 2.2.0 (2022-05-17)

- Implemented a better equation to calculate the mean radiant temperature

8.19 2.1.1 (2022-05-17)

- Fixed how DISC is calculated

8.20 2.1.0 (2022-04-20)

- Added Physiological Equivalent Temperature (PET) model
- In PMV and PPD function you can specify if occupants has control over airspeed

8.21 2.0.2 (2022-04-12)

- UTCI accepts lists as inputs

8.22 2.0.0 (2022-04-07)

Warning: Version 2.0.0 introduces some breaking changes. Now the default behaviour of most of the function is that they return a `np.nan` if the inputs are outside the model applicability limits.

For most functions we are no longer printing `Warnings`. If you want the function to return a value even if your inputs are outside the model applicability limits then you can set the variable `limit_input = False`. Please note that you should refrain from doing this.

Note: Starting from Version 2.0.0 of pythermalcomfort now most of the functions (see detailed list below) accept Numpy arrays or lists as inputs. This allows you to write more concise and faster code since we optimized vectorization, where possible using Numba.

- Allowing users to pass Numpy arrays or lists as input to the pmv_ppd, pmv, clo_tout, both adaptive models, utci, set_tmp, two_nodes
- Changed the input variable from return_invalid to limit_input
- Increased speed by using Numba @vectorize decorator
- Changed ASHRAE 55 2020 limits to match new addenda
- Improved documentation

8.23 1.11.0 (2022-03-16)

- Allowing users to pass a Numpy array as input into the UTCI function
- Numpy is now a requirement of pythermalcomfort
- Improved PMV, JOS-3, and UTCI documentation
- Testing PMV, SET, and solar gains models using online reference tables

8.24 1.10.0 (2021-11-15)

- Added JOS-3 model

8.25 1.9.0 (2021-10-07)

- Added Normal Effective Temperature (NET)
- Added Apparent Temperature (AT)
- Added Wind Chill Index (WCI)

8.26 1.8.0 (2021-09-28)

- Gagge's two-node model
- Added WBGT equation
- Added Heat index (HI)
- Added humidex index

8.27 1.7.1 (2021-09-08)

- Added ASHRAE equation to calculate the operative temperature

8.28 1.7.0 (2021-07-29)

- Implemented function to calculate the if fans are beneficial during heatwaves
- Fixed error in the SET equation to calculated radiative heat transfer coefficient
- Fixed error in SET definition
- Moved functions optimized with Numba to new file

8.29 1.6.2 (2021-07-08)

- Updated equation clo_dynamic based on ANSI/ASHRAE Addendum f to ANSI/ASHRAE Standard 55-2020
- Fixed import errors in examples

8.30 1.6.1 (2021-07-05)

- optimized UTCI function with Numba

8.31 1.6.0 (2021-05-21)

- (BREAKING CHANGE) moved some of the functions from psychrometrics to utilities
- added equation to calculate body surface area

8.32 1.5.2 (2021-05-05)

- return stress category UTCI

8.33 1.5.1 (2021-04-29)

- optimized phs with Numba

8.34 1.5.0 (2021-04-21)

- added Predicted Heat Strain (PHS) index from ISO 7933:2004

8.35 1.4.6 (2021-03-30)

- changed equation to calculate convective heat transfer coefficient in set_tmp() as per Gagge's 1986
- fixed vasodilation coefficient in set_tmp()
- docs changed term air velocity with air speed and improved documentation
- added new tests for comfort functions

8.36 1.3.6 (2021-02-04)

- fixed error calculation solar_altitude and sharp for supine person in solar_gain

8.37 1.3.5 (2021-02-02)

- not rounding SET temperature when calculating cooling effect

8.38 1.3.3 (2020-12-14)

- added function to calculate sky-vault view fraction

8.39 1.3.2 (2020-12-14)

- replaced input solar_azimuth with sharp in the solar_gain() function
- fixed small error in example pmv calculation

8.40 1.3.1 (2020-10-30)

- Fixed error calculation of cooling effect with elevated air temperatures

8.41 1.3.0 (2020-10-19)

- Changed PMV elevated air speed limit from 0.2 to 0.1 m/s

8.42 1.2.3 (2020-09-09)

- Fixed error in the calculation of erf
- Updated validation table erf

8.43 1.2.2 (2020-08-21)

- Changed default diameter in t_mrt
- Improved documentation

8.44 1.2.0 (2020-07-29)

- Significantly improved calculation speed using numba. Wrapped set and pmv functions

8.45 1.0.6 (2020-07-24)

- Minor speed improvement changed math.pow with **
- Added validation PMV validation table from ISO 7730

8.46 1.0.4 (2020-07-20)

- Improved speed calculation of the Cooling Effect
- Bisection has been replaced with Brentq function from scipy

8.47 1.0.3 (2020-07-01)

- Annotated variables in the SET code.

8.48 1.0.2 (2020-06-11)

- Fixed an error in the bisection equation used to calculated Cooling Effect.

8.49 1.0.0 (2020-06-09)

- Major stable release.

8.50 0.7.0 (2020-06-09)

- Added equation to calculate the dynamic clothing insulation

8.51 0.6.3 (2020-04-11)

- Fixed error in calculation adaptive ASHRAE
- Added some examples

8.52 0.6.3 (2020-03-17)

- Renamed function to_calc to t_o
- Fixed error calculation of relative air speed
- renamed input parameter ta to tdb
- Added function to calculate mean radiant temperature from black globe temperature
- Added function to calculate solar gain on people

- Added functions to calculate vapour pressure, wet-bulb temperature, dew point temperature, and psychrometric data from dry bulb temperature and RH
- Added authors
- Added dictionaries with reference clo and met values
- Added function to calculate enthalpy

8.53 0.5.2 (2020-03-11)

- Added function to calculate the running mean outdoor temperature

8.54 0.5.1 (2020-03-06)

- There was an error in version 0.4.2 in the calculation of PMV and PPD with elevated air speed, i.e. vr > 0.2 which has been fixed in this version
- Added function to calculate the cooling effect in accordance with ASHRAE

8.55 0.4.1 (2020-02-17)

- Removed compatibility with python 2.7 and 3.5

8.56 0.4.0 (2020-02-17)

- Created adaptive_EN, v_relative, t_clo, vertical_tmp_gradient, ankle_draft functions and wrote tests.
- Added possibility to decide with measuring system to use SI or IP.

8.57 0.3.0 (2020-02-13)

- Created set_tmp, adaptive_ashrae, UTCI functions and wrote tests.
- Added warning to let the user know if inputs entered do not comply with Standards applicability limits.

8.58 0.1.0 (2020-02-11)

- Created pmv, pmv_ppd functions and wrote tests.
- Documented code.

8.59 0.0.0 (2020-02-11)

- First release on PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX