Heat Transfer Documentation

Release 0.1.56

Caleb Bell

Aug 22, 2020
## 1 Tutorial

1.1 Introduction ................................................................. 3
1.2 Design philosophy .......................................................... 3
1.3 Insulation ................................................................. 4
1.4 Radiation ................................................................. 5
1.5 Heat exchanger sizing .................................................... 5
1.6 LMTD correction factor method ...................................... 5
1.7 Effectiveness-NTU method ............................................ 6

## 2 API Reference

2.1 Air cooler sizing and rating (ht.air_cooler) ......................... 9
2.2 Flow boiling (ht.boiling_flow) .................................... 20
2.3 Nucleic boiling and critical heat flux (ht.boiling_nucleic) .... 30
2.4 Boiling in plate and frame exchangers (ht.boiling_plate) ....... 44
2.5 Condensation (ht.condensation) .................................... 50
2.6 Conduction and shape factors (ht.conduction) ................... 55
2.7 External convection (ht.conv_external) .......................... 64
2.8 Free convection to immersed bodies (ht.conv_free_immersed) 74
2.9 Free convection to enclosed bodies (ht.conv_free_enclosed) ... 92
2.10 Internal convection (ht.conv_internal) .......................... 98
2.11 Convection to jacketed vessels (ht.conv_jacket) ................ 119
2.12 Convection to packed beds (ht.conv_packed_bed) ............ 123
2.13 Convection to Plate Heat Exchangers (single-phase) (ht.conv_plate) ........................................... 126
2.14 Convection with supercritical fluids (ht.conv_supercritical) .... 129
2.15 Heat transfer and pressure drop across tube bundles (ht.conv_tube_bank) ................. 144
2.16 Non boiling and non condensing two-phase heat transfer (ht.conv_two_phase) ............ 156
2.17 Miscellaneous utilities (ht.core) .................................... 166
2.18 Heat exchanger sizing and rating (ht.hx) ......................... 171
2.19 Database of insulating and refractory material properties (ht.insulation) ............. 218
2.20 Support for Numba (ht.numba) ..................................... 221
2.21 Heat transfer by radiation (ht.radiation) ........................ 223
2.22 Support for numpy arrays (ht.vectorized) ....................... 226

## 3 Installation ................................................................. 227

## 4 Latest source code ......................................................... 229
ht is open-source software for engineers and technicians working in the fields of chemical or mechanical engineering. It includes modules for various heat transfer functions.
1.1 Introduction

heat is the heat transfer component of the Chemical Engineering Design Library (ChEDL). Functions are provided to calculate heat transfer in a variety of situations, generally using dimensionless factors such as Reynolds and Prandtl number, and giving results in terms of dimensionless heat transfer coefficient, the Nusselt number. The ‘dimensional’ heat transfer coefficient may then be determined

\[ h = k \cdot \text{Nu}L \]

1.2 Design philosophy

Like all libraries, this was developed to scratch my own itches. Since its public release it has been found useful by many others, from students across the world to practicing engineers at some of the world’s largest companies.

The bulk of this library’s API is considered stable; enhancements to functions and classes will still happen, and default methods when using a generic correlation interface may change to newer and more accurate correlations as they are published and reviewed.

To the extent possible, correlations are implemented depending on the highest level parameters. The Nu_conv_internal correlation does not accept pipe diameter, velocity, viscosity, density, heat capacity, and thermal conductivity - it accepts Reynolds number and Prandtl number. This makes the API cleaner and encourages modular design.

All functions are designed to accept inputs in base SI units. However, any set of consistent units given to a function will return a consistent result; for instance, a function calculating volume doesn’t care if given an input in inches or meters; the output units will be the cube of those given to it. The user is directed to unit conversion libraries such as pint to perform unit conversions if they prefer not to work in SI units.

The standard math library is used in all functions except where special functions from numpy or scipy are necessary. SciPy is used for root finding, interpolation, scientific constants, ode integration, and its many special mathematical functions not present in the standard math library. The only other required library is the fluids library, a sister library for fluid dynamics. No other libraries will become required dependencies; anything else is optional.
There are two ways to use numpy arrays with ht. Easiest to use is a vectorized module, which wraps all of the ht functions with np.vectorize. Instead of importing from ht, the user can import from ht.vectorized:

```
>>> from ht.vectorized import *
>>> LMTD([100, 101], 60., 30., 40.2)
array([43.20040929, 43.60182765])
```

It is possible to switch back and forth between the namespaces with a subsequent import:

```
>>> from ht import *
```

The second way is Numba. This optional dependency provides the speed you expect from NumPy arrays - or better. In some cases, much better. The tutorial for using it is at ht.numba, but in general use it the same way but with a different import.

```
>>> from ht.numba_vectorized import *
```

### 1.3 Insulation

Insulating and refractory materials from the VDI Heat Transfer Handbook and the ASHRAE Handbook: Fundamentals have been digitized and are programatically available in ht. Density, heat capacity, and thermal conductivity are available although not all materials have all three.

The actual data is stored in a series of dictionaries, building_materials, ASHRAE_board_siding, ASHRAE_flooring, ASHRAE_insulation, ASHRAE_roofing, ASHRAE_plastering, ASHRAE_masonry, ASHRAE_woods, and refractories. A total of 390 different materials are available. Functions have been written to make accessing this data much more convenient.

To determine the correct string to look up a material by, one can use the function nearest_material:

```
>>> from ht import *
>>> nearest_material('stainless steel')
'Metals, stainless steel'
>>> nearest_material('mineral fibre')
'Mineral fiber'
```

Knowing a material’s ID, the functions k_material, rho_material, and Cp_material can be used to obtain its properties.

```
>>> wood = nearest_material('spruce')
>>> k_material(wood)
0.09
>>> rho_material(wood)
400.0
>>> Cp_material(wood)
1630.0
```

Materials which are refractories, stored in the dictionary refractories, have temperature dependent heat capacity and thermal conductivity between 400 °C and 1200 °C.

```
>>> C = nearest_material('graphite')
>>> k_material(C)
67.0
>>> k_material(C, T=800)
62.9851975
```

The limiting values are returned outside of this range:
1.4 Radiation

The Stefan-Boltzman law is implemented as $q_{\text{rad}}$. Optionally, a surrounding temperature may be specified as well. If the surrounding temperature is higher than the object, the calculated heat flux in W/m² will be negative, indicating the object is picking up heat not losing it.

```python
>>> q_rad(emissivity=1, T=400)
1451.613952
>>> q_rad(.85, T=400, T2=305.)
816.7821722650002
>>> q_rad(.85, T=400, T2=5000)  # ouch
-30122590.815640796
```

A blackbody’s spectral radiance can also be calculated, in units of W/steradian/square metre/metre. This calculation requires the temperature of the object and the wavelength to be considered.

```python
>>> blackbody_spectral_radiance(T=800., wavelength=4E-6)
1311694129.7430933
```

1.5 Heat exchanger sizing

There are three popular methods of sizing heat exchangers. The log-mean temperature difference correction factor method, the $\epsilon$-NTU method, and the P-NTU method. Each of those are cannot size a heat exchanger on their own - they do not care about heat transfer coefficients or area - but they must be used first to determine the thermal conditions of the heat exchanger. Sizing a heat exchanger is a very iterative process, and many designs should be attempted to determine the optimal one based on required performance and cost. The P-NTU method supports the most types of heat exchangers; its form always requires the UA term to be guessed however.

1.6 LMTD correction factor method

The simplest method, the log-mean temperature difference correction factor method, is as follows:

$$Q = U A \Delta T_{lm} F_t$$

Knowing the outlet and inlet temperatures of a heat exchanger and $Q$, one could determine $U A$ as follows:

```python
>>> dTlm = LMTD(Tci=15, Tco=85, Thi=130, Tho=110)
>>> Ft = F_LMTD_Fakheri(Tci=15, Tco=85, Thi=130, Tho=110, shells=1)
>>> Q = 1E6  # 1 MW
>>> UA = Q/(dTlm*Ft)
>>> UA
15833.566307803789
```

This method requires you to know all four temperatures before UA can be calculated. Fakheri developed a general expression for calculating $F_t$; it is valid for counterflow shell-and-tube exchangers with an even number of tube passes; the number of shell-side passes can be varied. $F_t$ is always less than 1, approaching 1 with very high numbers of shells.
No other expressions are available to calculate $F_l$ for different heat exchanger geometries; only the TEMA F and E exchanger types are really covered by this expression. However, with results from the other methods, $F_l$ can always be back-calculated.

Log mean temperature are available for both counterflow (by default) and co-current flow. This calculation does not depend on the units of temperature provided.

### 1.7 Effectiveness-NTU method

This method uses the formula $Q = \epsilon C_{\text{min}} (T_{hi,i} - T_{ci,i})$. The main complication of this method is calculating effectiveness $\epsilon$, which is a function of the mass flows, heat capacities, and $\text{UA} = f(NTU, Cr)$. The effectiveness-NTU method is implemented in $\text{effectiveness_from_NTU}$ and $\text{NTU_from_effectiveness}$. The supported heat exchanger types are somewhat limited; they are:

- Counterflow (ex. double-pipe)
- Parallel (ex. double pipe inefficient configuration)
- Shell and tube exchangers with even numbers of tube passes, one or more shells in series (TEMA E (one pass shell) only)
- Crossflow, single pass, fluids unmixed
- Crossflow, single pass, Cmax mixed, Cmin unmixed
- Crossflow, single pass, Cmin mixed, Cmax unmixed
- Boiler or condenser

To illustrate the method, first the individual methods will be used to determine the outlet temperatures of a heat exchanger. After, the more convenient and flexible wrapper $\text{effectiveness_NTU_method}$ is shown. Overall case of rating an existing heat exchanger where a known flowrate of steam and oil are contacted in crossflow, with the steam side mixed:
That was not very convenient. The more helpful wrapper `effectiveness_NTU_method` needs only the heat capacities and mass flows of each stream, the type of the heat exchanger, and one combination of the following inputs is required:

- Three of the four inlet and outlet stream temperatures
- Temperatures for the cold outlet and hot outlet and UA
- Temperatures for the cold inlet and hot inlet and UA
- Temperatures for the cold inlet and hot outlet and UA
- Temperatures for the cold outlet and hot inlet and UA

The function returns all calculated parameters for convenience as a dictionary.

Solve a heat exchanger to determine UA and effectiveness given the configuration, flows, subtype, the cold inlet/outlet temperatures, and the hot stream inlet temperature.

```python
>>> from pprint import pprint
>>> pprint(effectiveness_NTU_method(mh=5.2, mc=1.45, Cph=1860., Cpc=1900,
                      ... subtype='crossflow, mixed Cmax', Tci=15, Tco=85, Thi=130))
{'Cmax': 9672.0,
 'Cmin': 2755.0,
 'Cr': 0.2848428453267163,
 'NTU': 1.1040839095588,
 'Q': 192850.0,
 'Tci': 15,
 'Tco': 85,
 'Thi': 130,
 'Tho': 110.06100082712986,
 'UA': 3041.751170834494,
 'effectiveness': 0.6086956521739131}
```

Solve the same heat exchanger with the UA specified, and known inlet temperatures:

```python
>>> pprint(effectiveness_NTU_method(mh=5.2, mc=1.45, Cph=1860., Cpc=1900,
                      ... subtype='crossflow, mixed Cmax', Tci=15, Thi=130, UA=3041.75))
{'Cmax': 9672.0,
 'Cmin': 2755.0,
 'Cr': 0.2848428453267163,
 'NTU': 1.1040834845735028,
 'Q': 192849.96310220254,
 'Tci': 15,
 'Tco': 84.99998660697007,
 'Thi': 130,
 'Tho': 110.06100464203861,
 'UA': 3041.75,
 'effectiveness': 0.6086955357127832}
```
2.1 Air cooler sizing and rating (ht.air_cooler)

ht.air_cooler.Ft_aircooler (Thi, Tho, Tci, Tco, Ntp=1, rows=1)
Calculates log-mean temperature difference correction factor for a crossflow heat exchanger, as in an Air Cooler. Method presented in [1], fit to other’s nonexplicit work. Error is < 0.1%. Requires number of rows and tube passes as well as stream temperatures.

\[
F_T = 1 - \sum_{i=1}^{m} \sum_{k=1}^{n} a_{i,k} (1 - r_{1,m})^k \sin(2i \arctan R)
\]

\[
R = \frac{T_{hi} - T_{ho}}{T_{co} - T_{ci}}
\]

\[
r_{1,m} = \frac{\Delta T_{lm}}{T_{hi} - T_{ci}}
\]

Parameters
- Thi [float] Temperature of hot fluid in [K]
- Tho [float] Temperature of hot fluid out [K]
- Tci [float] Temperature of cold fluid in [K]
- Tco [float] Temperature of cold fluid out [K]
- Ntp [int] Number of passes the tubeside fluid will flow through [-]
- rows [int] Number of rows of tubes [-]

Returns
- Ft [float] Log-mean temperature difference correction factor [-]
Notes

This equation assumes that the hot fluid is tubside, as in the case of air coolers. The model is not symmetric, so ensure to switch around the inputs if using this function for other purposes.

This equation appears in [1]. It has been verified. For some cases, approximations are made to match coefficients with the number of tube passes and rows provided. 16 coefficients are used for each case; 8 cases are considered:

- 1 row 1 pass
- 2 rows 1 pass
- 2 rows 2 passes
- 3 rows 1 pass
- 3 rows 3 passes
- 4 rows 1 pass
- 4 rows 2 passes
- 4 rows 4 passes

References

[1]

Examples

```python
>>> Ft_aircooler(Thi=125., Tho=45., Tci=25., Tco=95., Ntp=1, rows=4)
0.5505093604092706
```

ht.air_cooler.air_coooler_noise_GPSA(tip_speed, power)
Calculates the noise generated by an air cooler bay with one fan according to the GPSA handbook [1].

\[
\text{PWL[dB(A)]} = 56 + 30 \log_{10} \left( \frac{\text{tip speed}[m/min]}{304.8[m/min]} \right) + 10 \log_{10}(\text{power}[hp])
\]

Parameters

- **tip_speed** [float] Tip speed of the air cooler fan blades, [m/s]
- **power** [float] Shaft power of single fan motor, [W]

Returns

- **noise** [float] Sound pressure level at 1 m from source, [dB(A)]

Notes

Internal units are in m/minute, and hp.

References

[1]
Examples

Example problem from GPSA [1].

```python
>>> air_cooler_noise_GPSA(tip_speed=3177/minute, power=25.1*hp)
100.53680477959792
```

**ht.air_cooler.air_cooler_noise_Mukherjee**(tip_speed, power, fan_diameter, induced=False)
Calculates the noise generated by an air cooler bay with one fan according to [1].

\[
\text{SPL\,[dB(A)]} = 46 + 30 \log_{10}(\text{tip speed})[m/s] + 10 \log_{10}(\text{power}[hp]) - 20 \log_{10}(D_{fan})
\]

Parameters

- **tip_speed** [float] Tip speed of the air cooler fan blades, [m/s]
- **power** [float] Shaft power of single fan motor, [W]
- **fan_diameter** [float] Diameter of air cooler fan, [m]
- **induced** [bool] Whether the air cooler is forced air (False) or induced air (True), [-]

Returns

- **noise** [float] Sound pressure level at 1 m from source (p0=2E-5 Pa), [dB(A)]

Notes

Internal units are in m/minute, hp, and m.

If the air cooler is induced, the sound pressure level is reduced by 3 dB.

References

[1]

Examples

```python
>>> air_cooler_noise_Mukherjee(tip_speed=3177/minute, power=25.1*hp, fan_diameter=4.267)
99.11026329092925
```

**ht.air_cooler.h_Briggs_Young**(m, A, A_min, A_increase, A_fin, ATube_showing, tube_diameter, fin_diameter, fin_thickness, bare_length, rho, Cp, nu, k, k_fin)
Calculates the air side heat transfer coefficient for an air cooler or other finned tube bundle with the formulas of Briggs and Young [1], [2] [3].

\[
Nu = 0.134Re^{0.681}Pr^{0.33} \left( \frac{S}{h} \right)^{0.2} \left( \frac{S}{b} \right)^{0.1134}
\]

Parameters

- **m** [float] Mass flow rate of air across the tube bank, [kg/s]
- **A** [float] Surface area of combined finned and non-finned area exposed for heat transfer, [m²]
- **A_min** [float] Minimum air flow area, [m²]

2.1. Air cooler sizing and rating (ht.air_cooler)
A_increase [float] Ratio of actual surface area to bare tube surface area 
\[ A_{increase} = \frac{A_{tube}}{A_{bare, total/tube}}, [-] \]

A_fin [float] Surface area of all fins in the bundle, [m^2]

A_tube_showing [float] Area of the bare tube which is exposed in the bundle, [m^2]

tube_diameter [float] Diameter of the bare tube, [m]

fin_diameter [float] Outer diameter of each tube after including the fin on both sides, [m]

fin_thickness [float] Thickness of the fins, [m]

bare_length [float] Length of bare tube between two fins bare length = fin interval – \( t_{fin} \), [m]

rho [float] Average (bulk) density of air across the tube bank, [kg/m^3]

Cp [float] Average (bulk) heat capacity of air across the tube bank, [J/kg/K]

mu [float] Average (bulk) viscosity of air across the tube bank, [Pa*s]

k [float] Average (bulk) thermal conductivity of air across the tube bank, [W/m/K]

k_fin [float] Thermal conductivity of the fin, [W/m/K]

Returns

h_bare_tube_basis [float] Air side heat transfer coefficient on a bare-tube surface area as if there were no fins present basis, [W/K/m^2]

Notes

The limits on this equation are 1000 < \( Re \) < 11.13 mm :math:`< D_o < 40.89 mm`, 1.42 mm < fin height < 16.57 mm, 0.33 mm < fin thickness < 2.02 mm, 1.30 mm < fin pitch < 4.06 mm, and 24.49 mm < normal pitch < 111 mm.

References

[1], [2], [3]

Examples

```python
>>> AC = AirCooledExchanger(tube_rows=4, tube_passes=4, tubes_per_row=20, tube_length=3,
... tube_diameter=1*inch, fin_thickness=0.000406, fin_density=1/0.002309,
... pitch_normal=.06033, pitch_parallel=.05207,
... fin_height=0.0159, tube_thickness=(.0254-.0186)/2,
... bundles_per_bay=1, parallel_bays=1, corbels=True)

>>> h_Briggs_Young(m=21.56, A=AC.A, A_min=AC.A_min, A_increase=AC.A_increase, A_fin=AC.A_fin,
... A_tube_showing=AC.A_tube_showing, tube_diameter=AC.tube_diameter,
... fin_diameter=AC.fin_diameter, bare_length=AC.bare_length,
... fin_thickness=AC.fin_thickness,
... rho=1.161, Cp=1007., mu=1.85E-5, k=0.0263, k_fin=205)
1422.872403237716
```
ht.air_cooler.h_ESDU_high_fin(m, A, A_min, A_increase, A_fin, A_tube_showing, tube_diameter, fin_diameter, fin_thickness, bare_length, pitch_parallel, pitch_normal, tube_rows, rho, Cp, mu, k, k_fin, Pr_wall=None)

Calculates the air side heat transfer coefficient for an air cooler or other finned tube bundle with the formulas of [2] as presented in [1].

\[
N_u = 0.242 Re^{0.658} \left( \frac{\text{bare length}}{\text{fin height}} \right)^{0.297} \left( \frac{P_1}{P_2} \right)^{-0.091} \rho^{3/3} \cdot F_1 \cdot F_2 
\]

\[
h_{A, total} = \frac{\eta A_{fin} + A_{bare, showing}}{A_{total}} h
\]

\[
h_{bare, total} = A_{increase} h_{A, total}
\]

**Parameters**

- **m** [float] Mass flow rate of air across the tube bank, [kg/s]
- **A** [float] Surface area of combined finned and non-finned area exposed for heat transfer, [m^2]
- **A_min** [float] Minimum air flow area, [m^2]
- **A_increase** [float] Ratio of actual surface area to bare tube surface area \( A_{increase} = \frac{A_{tube}}{A_{bare, total}/tube} \), [-]
- **A_fin** [float] Surface area of all fins in the bundle, [m^2]
- **A_tube_showing** [float] Area of the bare tube which is exposed in the bundle, [m^2]
- **tube_diameter** [float] Diameter of the bare tube, [m]
- **fin_diameter** [float] Outer diameter of each tube after including the fin on both sides, [m]
- **fin_thickness** [float] Thickness of the fins, [m]
- **bare_length** [float] Length of bare tube between two fins \( \text{bare length} = \text{fin interval} - t_{fin} \), [m]
- **pitch_parallel** [float] Distance between tube center along a line parallel to the flow; has been called longitudinal pitch, \( pp, s2, SL \), and \( p2 \), [m]
- **pitch_normal** [float] Distance between tube centers in a line 90° to the line of flow; has been called the transverse pitch, \( pn, s1, ST \), and \( p1 \), [m]
- **tube_rows** [int] Number of tube rows per bundle, [-]
- **rho** [float] Average (bulk) density of air across the tube bank, [kg/m^3]
- **Cp** [float] Average (bulk) heat capacity of air across the tube bank, [J/kg/K]
- **mu** [float] Average (bulk) viscosity of air across the tube bank, [Pa*s]
- **k** [float] Average (bulk) thermal conductivity of air across the tube bank, [W/m/K]
- **k_fin** [float] Thermal conductivity of the fin, [W/m/K]
- **Pr_wall** [float, optional] Prandtl number at the wall temperature; provide if a correction with the defaults parameters is desired; otherwise apply the correction elsewhere, [-]

**Returns**

- **h_bare_tube_basis** [float] Air side heat transfer coefficient on a bare-tube surface area as if there were no fins present basis, [W/K/m^2]
Notes

The tube-row count correction factor is 1 for four or more rows, 0.92 for three rows, 0.84 for two rows, and 0.76 for one row according to [1].

The property correction factor can be disabled by not specifying $Pr_{\text{wall}}$. A Prandtl number exponent of 0.26 is recommended in [1] for heating and cooling for both liquids and gases.

References

[1], [2], [3]

Examples

```python
>>> AC = AirCooledExchanger(tube_rows=4, tube_passes=4, tubes_per_row=20, tube_length=3,
... tube_diameter=1*inch, fin_thickness=0.000406, fin_density=1/0.002309,
... pitch_normal=0.06033, pitch_parallel=.05207,
... fin_height=0.0159, tube_thickness=(.0254-.0186)/2,
... bundles_per_bay=1, parallel_bays=1, corbels=True)

>>> h_ESDU_high_fin(m=21.56, A=AC.A, A_min=AC.A_min, A_increase=AC.A_increase, A_fin=AC.A_fin,
... A_tube_showing=AC.A_tube_showing, tube_diameter=AC.tube_diameter,
... fin_diameter=AC.fin_diameter, bare_length=AC.bare_length,
... fin_thickness=AC.fin_thickness, tube_rows=AC.tube_rows,
... pitch_normal=AC.pitch_normal, pitch_parallel=AC.pitch_parallel,
... rho=1.161, Cp=1007., mu=1.85E-5, k=0.0263, k_fin=205)

ht.air_cooler.h_ESDU_low_fin(m, A, A_min, A_increase, A_fin, A_tube_showing, tube_diameter, fin_diameter, fin_thickness, bare_length, pitch_parallel, pitch_normal, tube_rows, rho, Cp, mu, k, k_fin, Pr_wall=)  

Calculates the air side heat transfer coefficient for an air cooler or other finned tube bundle with low fins using the formulas of [1] as presented in [2] (and also [3]).

\[ N_u = 0.183Re^{0.7} \left( \frac{\text{bare length}}{\text{fin height}} \right)^{0.36} \left( \frac{p_1}{D_o} \right)^{0.06} \left( \frac{\text{fin height}}{D_o} \right)^{0.11} Pr^{0.36} \cdot F_1 \cdot F_2 \]

\[ h_{A,\text{total}} = \frac{\eta A_{\text{fin}} + A_{\text{bare,showing}} h}{A_{\text{total}}} \]

\[ h_{\text{bare, total}} = A_{\text{increase}} h_{A,\text{total}} \]

Parameters

- **m** [float] Mass flow rate of air across the tube bank, [kg/s]
- **A** [float] Surface area of combined finned and non-finned area exposed for heat transfer, [m²]
- **A_min** [float] Minimum air flow area, [m²]
- **A_increase** [float] Ratio of actual surface area to bare tube surface area $A_{\text{increase}} = \frac{A_{\text{total}}}{A_{\text{bare, total/tube}}}$, [-]
- **A_fin** [float] Surface area of all fins in the bundle, [m²]
- **A_tube_showing** [float] Area of the bare tube which is exposed in the bundle, [m²]
tube_diameter [float] Diameter of the bare tube, [m]
fin_diameter [float] Outer diameter of each tube after including the fin on both sides, [m]
fin_thickness [float] Thickness of the fins, [m]
bare_length [float] Length of bare tube between two fins bare length = fin interval − tf, [m]
pitch_parallel [float] Distance between tube center along a line parallel to the flow; has been called longitudinal pitch, pp, s2, SL, and p2, [m]
pitch_normal [float] Distance between tube centers in a line 90° to the line of flow; has been called the transverse pitch, pn, s1, ST, and p1, [m]
tube_rows [int] Number of tube rows per bundle, [-]
rho [float] Average (bulk) density of air across the tube bank, [kg/m^3]
Cp [float] Average (bulk) heat capacity of air across the tube bank, [J/kg/K]
mu [float] Average (bulk) viscosity of air across the tube bank, [Pa*s]
k [float] Average (bulk) thermal conductivity of air across the tube bank, [W/m/K]
k_fin [float] Thermal conductivity of the fin, [W/m/K]
Pr_wall [float, optional] Prandtl number at the wall temperature; provide if a correction with the defaults parameters is desired; otherwise apply the correction elsewhere, [-]

Returns

h_bare_tube_basis [float] Air side heat transfer coefficient on a bare-tube surface area as if there were no fins present basis, [W/K/m^2]

Notes

The tube-row count correction factor F2 can be disabled by setting tube_rows to 10. The property correction factor F1 can be disabled by not specifying Pr_wall. A Prandtl number exponent of 0.26 is recommended in [1] for heating and cooling for both liquids and gases.

There is a third correction factor in [1] for tube angles not 30, 45, or 60 degrees, but it is not fully explained and it is not shown in [2]. Another correction factor is in [2] for flow at an angle; however it would not make sense to apply it to finned tube banks due to the blockage by the fins.

References

[1], [2], [3]

Examples

```python
>>> AC = AirCooledExchanger(tube_rows=4, tube_passes=4, tubes_per_row=8, tube
˓→length=0.5,
... tube_diameter=0.0164, fin_thickness=0.001, fin_density=1/0.003,
... pitch_normal=0.0313, pitch_parallel=0.0271, fin_height=0.0041, corbels=True)

>>> h_ESDU_low_fin(m=0.914, A=AC.A, A_min=AC.A_min, A_increase=AC.A_increase, A_ ˓→fin=AC.A_fin,
... A_tube_showing=AC.A_tube_showing, tube_diameter=AC.tube_diameter,
... fin_diameter=AC.fin_diameter, bare_length=AC.bare_length,
```

(continues on next page)
ht.air_cooler.h_Ganguli_VDI(m, A, A_min, A_increase, A_fin, A_tube_showing, tube_diameter, fin_diameter, fin_thickness, bare_length, pitch_parallel, pitch_normal, tube_rows, rho, Cp, mu, k, k_fin)

Calculates the air side heat transfer coefficient for an air cooler or other finned tube bundle with the formulas of [1] as modified in [2].

Inline:

\[ N_u_d = 0.22 \frac{Re_d^{0.6}}{\left( \frac{A}{A_{tube, only}} \right)^{0.15} \sqrt[3]{P_r}} \]

Staggered:

\[ N_u_d = 0.38 \frac{Re_d^{0.6}}{\left( \frac{A}{A_{tube, only}} \right)^{0.15} \sqrt[3]{P_r}} \]

**Parameters**

- **m** [float] Mass flow rate of air across the tube bank, [kg/s]
- **A** [float] Surface area of combined finned and non-finned area exposed for heat transfer, [m^2]
- **A_min** [float] Minimum air flow area, [m^2]
- **A_increase** [float] Ratio of actual surface area to bare tube surface area

\[ A_{increase} = \frac{A}{A_{tube, only}} \frac{A_{tube, only}}{A_{bare, total/tube}} \]

- **A_fin** [float] Surface area of all fins in the bundle, [m^2]
- **A_tube_showing** [float] Area of the bare tube which is exposed in the bundle, [m^2]
- **tube_diameter** [float] Diameter of the bare tube, [m]
- **fin_diameter** [float] Outer diameter of each tube after including the fin on both sides, [m]
- **fin_thickness** [float] Thickness of the fins, [m]
- **bare_length** [float] Length of bare tube between two fins bare length = fin interval − \( t_fi n \), [m]
- **pitch_parallel** [float] Distance between tube center along a line parallel to the flow; has been called *longitudinal* pitch, \( pp, s_2, SL \), and \( p_2 \), [m]
- **pitch_normal** [float] Distance between tube centers in a line 90° to the line of flow; has been called the *transverse* pitch, \( pn, s_1, ST \), and \( p_1 \), [m]
- **tube_rows** [int] Number of tube rows per bundle, [-]
- **rho** [float] Average (bulk) density of air across the tube bank, [kg/m^3]
- **Cp** [float] Average (bulk) heat capacity of air across the tube bank, [J/kg/K]
- **mu** [float] Average (bulk) viscosity of air across the tube bank, [Pa*s]
- **k** [float] Average (bulk) thermal conductivity of air across the tube bank, [W/m/K]
- **k_fin** [float] Thermal conductivity of the fin, [W/m/K]

**Returns**

- **h_bare_tube_basis** [float] Air side heat transfer coefficient on a bare-tube surface area as if there were no fins present basis, [W/K/m^2]
Notes

The VDI modifications were developed in comparison with HTFS and HTRI data according to [2].

For cases where the tube row count is less than four, the coefficients are modified in [2]. For the inline case, 0.2 replaces 0.22. For the staggered cases, the coefficient is 0.2, 0.33, 0.36 for 1, 2, or 3 tube rows respectively.

The model is also shown in [4].

References

[1], [2], [3], [4]

Examples

Example 12.1 in [3]:

```python
>>> AC = AirCooledExchanger(tube_rows=4, tube_passes=4, tubes_per_row=56, tube_length=36*ft, tube_diameter=1*inch, fin_thickness=0.013*inch, fin_density=10/inch, angle=30, pitch_normal=2.5*inch, fin_height=0.625*inch, corbels=True)
```

```python
>>> h_Ganguli_VDI(m=130.70315, A=AC.A, A_min=AC.A_min, A_increase=AC.A_increase, A_fin=AC.A_fin, ... A_tube_showing=AC.A_tube_showing, tube_diameter=AC.tube_diameter, ... fin_diameter=AC.fin_diameter, bare_length=AC.bare_length, ... fin_thickness=AC.fin_thickness, tube_rows=AC.tube_rows, ... pitch_parallel=AC.pitch_parallel, pitch_normal=AC.pitch_normal, ... rho=1.2013848, Cp=1009.0188, mu=1.9304793e-05, k=0.027864828, k_fin=238)
969.2850818578595
```

ht.air_cooler.dP_ESDU_high_fin(m, A_min, A_increase, flow_area_contraction_ratio, tube_diameter, pitch_parallel, pitch_normal, tube_rows, rho, mu)

Calculates the air-side pressure drop for a high-finned tube bank according to the ESDU [1] method, as described in [2]. This includes the effects of friction of the fin, and acceleration.

\[
\Delta P = (K_{acc} + n_{rows}K_f) \frac{1}{2} \rho v_{max}^2
\]

\[
K_f = 4.567 Re_D^{-0.242} \left( \frac{A}{A_{tube,only}} \right)^{0.504} \left( \frac{p_1}{D_o} \right)^{-0.376} \left( \frac{p_2}{D_o} \right)^{-0.546}
\]

\[
K_{acc} = 1 + (\text{flow area contraction ratio})^2
\]

Parameters

- `m` [float] Mass flow rate of air across the tube bank, [kg/s]
- `A_min` [float] Minimum air flow area, [m^2]
- `A_increase` [float] Ratio of actual surface area to bare tube surface area \(A_{increase} = \frac{A_{tube}}{A_{bare,total/tube}}\), [-]
- `flow_area_contraction_ratio` [float] Ratio of `A_min` to `A_face`, [-]
- `tube_diameter` [float] Diameter of the bare tube, [m]

2.1. Air cooler sizing and rating (ht.air_cooler)
**pitch_parallel** [float] Distance between tube center along a line parallel to the flow; has been called *longitudinal* pitch, *pp*, *s2*, *SL*, and *p2*, [m]

**pitch_normal** [float] Distance between tube centers in a line 90° to the line of flow; has been called the *transverse* pitch, *pn*, *s1*, *ST*, and *p1*, [m]

**tube_rows** [int] Number of tube rows per bundle, [-]

**rho** [float] Average (bulk) density of air across the tube bank, [kg/m³]

**mu** [float] Average (bulk) viscosity of air across the tube bank, [Pa*s]

**Returns**

**dP** [float] Overall pressure drop across the finned tube bank, [Pa]

**Notes**

The data used by the ESDU covered:

- fin density 4 to 11/inch
- tube outer diameters 3/8 to 2 inches
- fin heights 1/3 to 5/8 inches
- fin tip to fin root diameters 1.2 to 2.4
- Reynolds numbers 5000 to 50000

[1] claims 72% of experimental points were within 10% of the results of the correlation.

The Reynolds number used in this equation is that based on *V_max*, calculated using the minimum flow area.

**References**

[1], [2]

**Examples**

```python
>>> AC = AirCooledExchanger(tube_rows=4, tube_passes=4, tubes_per_row=8, tube_length=0.5,
... tube_diameter=0.0164, fin_thickness=0.001, fin_density=1/0.003,
... pitch_normal=0.0313, pitch_parallel=0.0271, fin_height=0.0041, corbels=True)

485.6307687791502
```

```python
>>> dP_ESDU_high_fin(m=0.914, A_min=AC.A_min, A_increase=AC.A_increase, flow_area_contraction_ratio=AC.flow_area_contraction_ratio, tube_diameter=AC.tube_diameter, pitch_parallel=AC.pitch_parallel, pitch_normal=AC.pitch_normal, tube_rows=AC.tube_rows, rho=1.217, mu=0.000018)
485.6307687791502
```

ht.air_cooler.dP_ESDU_low_fin(m, A_min, A_increase, flow_area_contraction_ratio, tube_diameter, fin_height, bare_length, pitch_parallel, pitch_normal, tube_rows, rho, mu)

Calculates the air-side pressure drop for a low-finned tube bank according to the ESDU [1] method, as described in [2]. This includes the effects of friction of the fin, and acceleration.

$$\Delta P = (K_{acc} + n_{rows}K_f)\frac{1}{2}\rho v_{max}^2$$
Heat Transfer Documentation, Release 0.1.56

\[ K_f = 4.71 R e_D^{-0.286} \left( \frac{\text{fin height}}{\text{bare length}} \right)^{0.51} \left( \frac{p_1 - D_o}{p_2 - D_o} \right)^{0.536} \left( \frac{D_o}{p_1 - D_o} \right)^{0.36} \]

\[ K_{acc} = 1 + (\text{flow area contraction ratio})^2 \]

Parameters

- \( m \) [float] Mass flow rate of air across the tube bank, [kg/s]
- \( A_{\text{min}} \) [float] Minimum air flow area, [m²]
- \( A_{\text{increase}} \) [float] Ratio of actual surface area to bare tube surface area \( A_{\text{increase}} = \frac{A_{\text{tube}}}{A_{\text{bare, total/tube}}} \), [-]
- \( \text{flow_area_contraction_ratio} \) [float] Ratio of \( A_{\text{min}} \) to \( A_{\text{face}} \), [-]
- \( \text{tube_diameter} \) [float] Diameter of the bare tube, [m]
- \( \text{fin_height} \) [float] Height above bare tube of the tube fins, [m]
- \( \text{bare_length} \) [float] Length of bare tube between two fins bare length = fin interval - \( t_f i n \), [m]
- \( \text{pitch_parallel} \) [float] Distance between tube center along a line parallel to the flow; has been called longitudinal pitch, \( pp, s_2, SL \), and \( p_2 \), [m]
- \( \text{pitch_normal} \) [float] Distance between tube centers in a line 90° to the line of flow; has been called the transverse pitch, \( pn, s_1, ST \), and \( p_1 \), [m]
- \( \text{tube_rows} \) [int] Number of tube rows per bundle, [-]
- \( \rho \) [float] Average (bulk) density of air across the tube bank, [kg/m³]
- \( \mu \) [float] Average (bulk) viscosity of air across the tube bank, [Pa*s]

Returns

- \( dP \) [float] Overall pressure drop across the finned tube bank, [Pa]

Notes

Low fins are fins which were formed on the tube outside wall, normally by the cold rolling process. The data used by the ESDU covered:

- fin density 11 to 32/inch
- tube outer diameters 0.5 to 1.25 inches
- fin heights 0.03 to 0.1 inches
- Reynolds numbers 1000 to 80000

[1] compared this correlation with 81 results and obtained a standard deviation of 7.7%.

The Reynolds number used in this equation is that based on \( V_{\text{max}} \), calculated using the minimum flow area.

References

[1], [2]

Examples

2.1. Air cooler sizing and rating (ht.air_cooler) 19
>>> AC = AirCooledExchanger(tube_rows=4, tube_passes=4, tubes_per_row=8, tube_length=0.5, ...
tube_diameter=0.0164, fin_thickness=0.001, fin_density=1/0.003, ...
pitch_normal=0.0313, pitch_parallel=0.0271, fin_height=0.0041, corbels=True)

>>> dP_ESDU_low_fin(m=0.914, A_min=AC.A_min, A_increase=AC.A_increase, ...
flow_area_contraction_ratio=AC.flow_area_contraction_ratio, ...
tube_diameter=AC.tube_diameter, fin_height=AC.fin_height, ...
bare_length=AC.bare_length, pitch_parallel=AC.pitch_parallel, ...
pitch_normal=AC.pitch_normal, tube_rows=AC.tube_rows, rho=1.217, ...
mu=0.000018)

464.54331418655914

2.2 Flow boiling (ht.boiling_flow)

ht.boiling_flow Thome (m, x, D, rho, rhog, mul, kl, kg, Cpl, Cpg, Hvap, sigma, Psat, Pc, q=None, Te=None)
Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is as developed in [1] and [2], and also reviewed [3]. This is a complicated model, but expected to have more accuracy as a result.

Either the heat flux or excess temperature is required for the calculation of heat transfer coefficient. The solution for a specified excess temperature is solved numerically, making it slow.

\[
\begin{align*}
    h(z) &= \frac{t_l}{\tau} h_l(z) + \frac{t_{film}}{\tau} h_{film}(z) + \frac{t_{dry}}{\tau} h_g(z) \\
    h_{l/g}(z) &= (Nu_{laminar} + Nu_{trans})^{1/4} k/D \\
    Nu_{laminar} &= 0.91 Pr^{1/3} \sqrt{ReD/L(z)} \\
    Nu_{trans} &= \frac{(f/8)(Re - 1000)Pr}{1 + 12.7(f/8)^{1/2}(Pr^{2/3} - 1)} \left[ 1 + \left( \frac{D}{L(z)} \right)^{2/3} \right] \\
    f &= (1.82 \log_{10} Re - 1.64)^{-2} \\
    L_l &= \frac{xG_{top}}{\rho_l} (1 - x) \\
    L_{dry} &= \frac{\tau}{\rho_l \rho_g} t_{dry} \\
    t_l &= \frac{\tau}{1 + \frac{\rho_l}{\rho_g} x} \\
    t_v &= \frac{\tau}{1 + \frac{\rho_l}{\rho_v} (1-x)} \\
    \tau &= \frac{1}{f_{opt}} \\
    f_{opt} &= \left( \frac{q}{q_{ref}} \right)^{n_f} \\
    q_{ref} &= 3328 \left( \frac{P_{sat}}{P_c} \right)^{-0.5} \\
    t_{dry, film} &= \frac{\rho_l \Delta Hvap}{q} [\delta_l(z) - \delta_{min}] 
\end{align*}
\]
\[
\frac{\delta_0}{D} = C_{\delta_0} \left(3 \sqrt{\frac{\mu_l}{v_pD}}\right)^{0.84} \left[(0.07Bo^{0.41})^{-8} + 0.1^{-8}\right]^{-1/8}
\]

\[
Bo = \frac{\rho_lD}{\sigma v_p^2}
\]

\[
v_p = G_{tp} \left[\frac{x}{\rho_g} + \frac{1-x}{\rho_l}\right]
\]

\[
h_{film}(z) = \frac{2k_l}{\delta_0(z) + \delta_{min}(z)}
\]

\[
\delta_{min} = 0.3 \cdot 10^{-6} m
\]

\[
C_{\delta_0} = 0.29
\]

\[
n_f = 1.74
\]

if t dry film > tv:

\[
\delta_{end}(x) = \delta(z, t_v)
\]

\[
t_{film} = t_v
\]

\[
t_{dry} = 0
\]

Otherwise:

\[
\delta_{end}(z) = \delta_{min}
\]

\[
t_{film} = t_{dry, film}
\]

\[
t_{dry} = t_v - t_{film}
\]

**Parameters**

- \textbf{m} [float] Mass flow rate [kg/s]
- \textbf{x} [float] Quality at the specific tube interval []
- \textbf{D} [float] Diameter of the tube [m]
- \textbf{rhol} [float] Density of the liquid [kg/m^3]
- \textbf{rhog} [float] Density of the gas [kg/m^3]
- \textbf{mul} [float] Viscosity of liquid [Pa*s]
- \textbf{mug} [float] Viscosity of gas [Pa*s]
- \textbf{kl} [float] Thermal conductivity of liquid [W/m/K]
- \textbf{kg} [float] Thermal conductivity of gas [W/m/K]
- \textbf{Cpl} [float] Heat capacity of liquid [J/kg/K]
- \textbf{Cpg} [float] Heat capacity of gas [J/kg/K]
- \textbf{Hvap} [float] Heat of vaporization of liquid [J/kg]
- \textbf{sigma} [float] Surface tension of liquid [N/m]
- \textbf{Psat} [float] Vapor pressure of fluid, [Pa]
- \textbf{Pc} [float] Critical pressure of fluid, [Pa]
- \textbf{q} [float, optional] Heat flux to wall [W/m^2]
**Te** [float, optional] Excess temperature of wall, [K]

**Returns**

- **h** [float] Heat transfer coefficient [W/m²/K]

**Notes**

[1] and [2] have been reviewed, and are accurately reproduced in [3].

[1] used data from 7 studies, covering 7 fluids and Dh from 0.7-3.1 mm, heat flux from 0.5-17.8 W/cm², x from 0.01-0.99, and G from 50-564 kg/m²/s.

Liquid and/or gas slugs are both considered, and are hydrodynamically developing. \( L_l \) is the calculated length of liquid slugs, and \( L_{dry} \) is the same for vapor slugs.

Because of the complexity of the model and that there is some logic in this function, \( Te \) as an input may lead to a different solution that the calculated \( q \) will in return.

**References**

[1], [2], [3]

**Examples**

```python
>>> Thome(m=1, x=0.4, D=0.3, rhol=567., rhog=18.09, kl=0.086, kg=0.2,
...   mul=156E-6, mug=1E-5, Cpl=2300, Cpg=1400, sigma=0.02, Hvap=9E5,
...   Psat=1E5, Pc=22E6, q=1E5)
1633.008836502032
```

ht.boiling_flow.Liu_Winterton \((m, x, D, rhol, rhog, mul, kl, Cpl, MW, P, Pc, Te)\)

Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is as developed in [1], also reviewed in [2] and [3].

Excess wall temperature is required to use this correlation.

\[
\begin{align*}
\mathbf{h}_{tp} &= \sqrt{(F \cdot h_t)^2 + (S \cdot h_{nb})^2} \\
\cdot \mathbf{math} &\quad S = \left(1 + 0.055 F^{0.1} Re_L^{0.16}\right)^{-1} \\
\cdot \mathbf{math} &\quad h_t = 0.023 Re_L^{0.8} Pr_l^{0.4} k_l / D \\
\cdot \mathbf{math} &\quad Re_L = \frac{GD}{\mu_l} \\
\cdot \mathbf{math} &\quad F = \left[1 + x Pr_l (\rho_l/\rho_g - 1)^{0.35}\right]^{-1.33}
\end{align*}
\]

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific tube interval []
- **D** [float] Diameter of the tube [m]
- **rhol** [float] Density of the liquid [kg/m³]
rhog [float] Density of the gas [kg/m^3]
mul [float] Viscosity of liquid [Pa*s]
kl [float] Thermal conductivity of liquid [W/m/K]
Cpl [float] Heat capacity of liquid [J/kg/K]
MW [float] Molecular weight of the fluid, [g/mol]
P [float] Pressure of fluid, [Pa]
Pc [float] Critical pressure of fluid, [Pa]
Te [float, optional] Excess temperature of wall, [K]

Returns

h [float] Heat transfer coefficient [W/m^2/K]

Notes

[1] has been reviewed, and is accurately reproduced in [3].

Uses the Cooper and turbulent_Dittus_Boelter correlations.

A correction for horizontal flow at low Froude numbers is available in [1] but has not been implemented and is not recommended in several sources.

References

[1], [2], [3]

Examples

```
>>> Liu_Winterton(m=1, x=0.4, D=0.3, rhol=567., rhog=18.09, kl=0.086, 
... mul=156E-6, Cpl=2300, P=1E6, Pc=22E6, MW=44.02, Te=7)
4747.749477190532
```

ht.boiling_flow.Chen_Edelstein (m, x, D, rhol, rhog, mul, mug, kl, Cpl, Hvap, sigma, dPsat, Te)
Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is developed in [1] and [2], and reviewed in [3]. This model is one of the most often used. It uses the Dittus-Boelter correlation for turbulent convection and the Forster-Zuber correlation for pool boiling, and combines them with two factors F and S.

\[
h_{tp} = S \cdot h_{nb} + F \cdot h_{sp,l}
\]

\[
h_{sp,l} = 0.023Re_l^{0.8}Pr_l^{0.4}kl/D
\]

\[
Re_l = \frac{DG(1-x)}{\mu_l}
\]

\[
h_{nb} = 0.00122 \left( \frac{\lambda_l^{0.79}c_{pl}^{0.45}T_l^{0.49}t_{vap}^{0.49}P_l^{0.24}P_g^{0.24}}{\sigma^{0.5}\mu_l^{0.29}T_l^{0.24}t_{vap}^{0.24}P_l^{0.24}} \right) \Delta T_{sat}^{0.24} \Delta P_{sat}^{0.75}
\]

\[
F = (1 + X_{lt}^{-0.5})^{1.78}
\]

\[
X_{lt} = \left( \frac{1-x}{x} \right)^{0.9} \left( \frac{\rho_g}{\rho_l} \right)^{0.5} \left( \frac{\mu_l}{\mu_g} \right)^{0.1}
\]

2.2. Flow boiling (ht.boiling_flow)
\[ S = 0.9622 - 0.5822 \left( \tan^{-1} \left( \frac{Re_L \cdot F^{1.25}}{6.18 \cdot 10^4} \right) \right) \]

**Parameters**

- \( m \) [float] Mass flow rate [kg/s]
- \( x \) [float] Quality at the specific tube interval []
- \( D \) [float] Diameter of the tube [m]
- \( \rho_l \) [float] Density of the liquid [kg/m³]
- \( \rho_g \) [float] Density of the gas [kg/m³]
- \( \mu_l \) [float] Viscosity of liquid [Pa*s]
- \( \mu_g \) [float] Viscosity of gas [Pa*s]
- \( k_l \) [float] Thermal conductivity of liquid [W/m/K]
- \( C_{pl} \) [float] Heat capacity of liquid [J/kg/K]
- \( H_{vap} \) [float] Heat of vaporization of liquid [J/kg]
- \( \sigma \) [float] Surface tension of liquid [N/m]
- \( d_{Psat} \) [float] Difference in Saturation pressure of fluid at \( Te \) and \( T \), [Pa]
- \( Te \) [float] Excess temperature of wall, [K]

**Returns**

- \( h \) [float] Heat transfer coefficient [W/m²/K]

**See also:**

- *turbulent_Dittus_Boelter*
- *Forster_Zuber*

**Notes**

[1] and [2] have been reviewed, but the model is only put together in the review of [3]. Many other forms of this equation exist with different functions for \( F \) and \( S \).

**References**

[1], [2], [3]

**Examples**

```python
>>> Chen_Edelstein(m=0.106, x=0.2, D=0.0212, rho_l=567, rho_g=18.09,
                  ... mul=156E-6, mug=7.11E-6, kl=0.086, Cpl=2730, Hvap=2E5, sigma=0.02,
                  ... dPsat=1E5, Te=3)
3289.058731974052
```

ht.boiling_flow.Chen_Bennett(\( m, x, D, \rho_l, \rho_g, \mu_l, \mu_g, k_l, C_{pl}, H_{vap}, \sigma, d_{Psat}, T_e \))

Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is developed in [1] and [2], and reviewed in [3]. This model is one of the most often used, and replaces the
Chen_Edelstein correlation. It uses the Dittus-Boelter correlation for turbulent convection and the Forster-Zuber correlation for pool boiling, and combines them with two factors $F$ and $S$.

$$h_{tp} = S \cdot h_{nb} + F \cdot h_{sp,l}$$

..math:: h_{sp,l} = 0.023Re_l^{0.8}Pr_l^{0.4}k_l/D$$

..math:: Re_l = \frac{DG(1-x)}{\mu_l}$$

..math:: h_{nb} = 0.00122 \left( \frac{\lambda_{l,0.79}^{0.45} \rho_{l,0.49}^{0.29} H_{vap,0.24}^{0.24}}{\sigma^{0.5} \mu^{0.29} \rho^{0.49} \rho_l^{0.24}} \right) \Delta T_{sat}^{0.24} \Delta p_{sat}^{0.75}$$

..math:: F = \left( \frac{Pr_l + 1}{2} \right)^{0.444} \cdot (1 + X_{tt}^{-0.5})^{1.78}$$

..math:: S = 1 - \exp\left( -F \cdot h_{conv} \cdot X_0 / k_l \right)$$

..math:: X_{tt} = \left( 1 - \frac{x}{x} \right) \left( \frac{\rho_g}{\rho_l} \right)^{0.9} \left( \frac{\mu_g}{\mu_l} \right)^{0.1}$$

..math:: X_0 = 0.041 \left( \frac{\sigma}{g \cdot (\rho_l - \rho_v)} \right)^{0.5}$$

Parameters

- m [float] Mass flow rate [kg/s]
- x [float] Quality at the specific tube interval []
- D [float] Diameter of the tube [m]
- rhol [float] Density of the liquid [kg/m^3]
- rhog [float] Density of the gas [kg/m^3]
- mul [float] Viscosity of liquid [Pa*s]
- mug [float] Viscosity of gas [Pa*s]
- kl [float] Thermal conductivity of liquid [W/m/K]
- Cpl [float] Heat capacity of liquid [J/kg/K]
- Hvap [float] Heat of vaporization of liquid [J/kg]
- sigma [float] Surface tension of liquid [N/m]
- dPsat [float] Difference in Saturation pressure of fluid at Te and T, [Pa]
- Te [float] Excess temperature of wall, [K]

Returns

- h [float] Heat transfer coefficient [W/m^2/K]

See also:

Chen_Edelstein

turbulent_Dittus_Boelter

Forster_Zuber

---

2.2. Flow boiling (ht.boiling_flow)
Notes

[1] and [2] have been reviewed, but the model is only put together in the review of [3]. Many other forms of this equation exist with different functions for $F$ and $S$.

References

[1], [2], [3]

Examples

```python
>>> Chen_Bennett(m=0.106, x=0.2, D=0.0212, rhol=567, rhog=18.09,
... mul=156E-6, mug=7.11E-6, kl=0.086, Cpl=2730, Hvap=2E5, sigma=0.02,
... dPsat=1E5, Te=3)
4938.275351219369
```

ht.boiling_flow.Lazarek_Black($m$, $D$, $mul$, $kl$, $Hvap$, $q=None$, $Te=None$)
Calculates heat transfer coefficient for film boiling of saturated fluid in vertical tubes for either upward or downward flow. Correlation is as shown in [1], and also reviewed in [2] and [3].

Either the heat flux or excess temperature is required for the calculation of heat transfer coefficient.

Quality independent. Requires no properties of the gas. Uses a Reynolds number assuming all the flow is liquid.

$$h_{tp} = 30Re_{lo}^{0.857}Bg^{0.714}k_{l}$$

$$Re_{lo} = \frac{G_{tp}D}{\mu_{l}}$$

Parameters

- $m$ [float] Mass flow rate [kg/s]
- $D$ [float] Diameter of the channel [m]
- $mul$ [float] Viscosity of liquid [Pa*s]
- $kl$ [float] Thermal conductivity of liquid [W/m/K]
- $Hvap$ [float] Heat of vaporization of liquid [J/kg]
- $q$ [float, optional] Heat flux to wall [W/m^2]
- $Te$ [float, optional] Excess temperature of wall, [K]

Returns

- $h$ [float] Heat transfer coefficient [W/m^2/K]

Notes

[1] has been reviewed.

[2] claims it was developed for a range of quality 0-0.6, Relo 860-5500, mass flux 125-750 kg/m^2/s, q of 1.4-38 W/cm^2, and with a pipe diameter of 3.1 mm. Developed with data for R113 only.
References

[1], [2], [3]

Examples

```python
>>> Lazarek_Black(m=10, D=0.3, mul=1E-3, kl=0.6, Hvap=2E6, Te=100)
9501.932636079293
```

ht.boiling_flow. Li_Wu (m, x, D, rhol, rhog, mul, kl, Hvap, sigma, q=None, Te=None)

Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is as shown in [1], and also reviewed in [2] and [3].

Either the heat flux or excess temperature is required for the calculation of heat transfer coefficient. Uses liquid Reynolds number, Bond number, and Boiling number.

\[
h_{tp} = 334 Bg^{0.3} (Bo \cdot Re_l^{0.36})^{0.4} \frac{kl}{D}
\]

\[
Re_l = \frac{G(1-x)D}{\mu_l}
\]

Parameters

- `m` [float] Mass flow rate [kg/s]
- `x` [float] Quality at the specific tube interval []
- `D` [float] Diameter of the tube [m]
- `rhol` [float] Density of the liquid [kg/m^3]
- `rhog` [float] Density of the gas [kg/m^3]
- `mul` [float] Viscosity of liquid [Pa*s]
- `kl` [float] Thermal conductivity of liquid [W/m/K]
- `Hvap` [float] Heat of vaporization of liquid [J/kg]
- `sigma` [float] Surface tension of liquid [N/m]
- `q` [float, optional] Heat flux to wall [W/m^2]
- `Te` [float, optional] Excess temperature of wall, [K]

Returns

- `h` [float] Heat transfer coefficient [W/m^2/K]

Notes

[1] has been reviewed.

[1] used 18 sets of experimental data to derive the results, covering hydraulic diameters from 0.19 to 3.1 mm and 12 different fluids.

References

[1], [2], [3]
Examples

```python
>>> Li_Wu(m=1, x=0.2, D=0.3, rhol=567., rhog=18.09, kl=0.086, mul=156E-6, sigma=0.02, Hvap=9E5, q=1E5)
5345.409399239492
```

ht.boiling_flow.Sun_Mishima (m, D, rhol, rhog, mul, kl, Hvap, sigma, q=None, Te=None)
Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is as shown in [1], and also reviewed in [2] and [3].

Either the heat flux or excess temperature is required for the calculation of heat transfer coefficient. Uses liquid-only Reynolds number, Weber number, and Boiling number. Weber number is defined in terms of the velocity if all fluid were liquid.

\[
h_{tp} = \frac{6Re_{lo}^{1.05} B_g^{0.54}}{We_l^{0.191} (\rho_l/\rho_g)^{0.142}} \frac{k_l}{D}
\]

\[
Re_{lo} = \frac{G_{tp} D}{\mu_l}
\]

Parameters
- **m** [float] Mass flow rate [kg/s]
- **D** [float] Diameter of the tube [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **mul** [float] Viscosity of liquid [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Hvap** [float] Heat of vaporization of liquid [J/kg]
- **sigma** [float] Surface tension of liquid [N/m]
- **q** [float, optional] Heat flux to wall [W/m^2]
- **Te** [float, optional] Excess temperature of wall, [K]

Returns
- **h** [float] Heat transfer coefficient [W/m^2/K]

Notes
[1] has been reviewed.
[1] used 2501 data points to derive the results, covering hydraulic diameters from 0.21 to 6.05 mm and 11 different fluids.

References
[1], [2]
Examples

```python
>>> Sun_Mishima(m=1, D=0.3, rhol=567., rhog=18.09, kl=0.086, mul=156E-6, sigma=0.
˓→02, Hvap=9E5, Te=10)
507.6709168372167
```

Calculates heat transfer coefficient for film boiling of saturated fluid in any orientation of flow. Correlation is as shown in [1] and [2], and also reviewed in [3].

Either the heat flux or excess temperature is required for the calculation of heat transfer coefficient. Uses liquid Reynolds number, Weber number, and Boiling number. Weber number is defined in terms of the velocity if all fluid were liquid.

\[
h_{tp} = 136876(Bg \cdot W_{el})^{0.1993} Re_{el}^{-0.1626}
\]

\[
Re_{el} = \frac{GD(1-x)}{\mu_l}
\]

\[
W_{el} = \frac{G^2 D}{\rho_l \sigma}
\]

Parameters

- \(m\) [float] Mass flow rate [kg/s]
- \(x\) [float] Quality at the specific tube interval []
- \(D\) [float] Diameter of the tube [m]
- \(rhol\) [float] Density of the liquid [kg/m^3]
- \(mul\) [float] Viscosity of liquid [Pa*s]
- \(Hvap\) [float] Heat of vaporization of liquid [J/kg]
- \(sigma\) [float] Surface tension of liquid [N/m]
- \(q\) [float, optional] Heat flux to wall [W/m^2]
- \(Te\) [float, optional] Excess temperature of wall, [K]

Returns

- \(h\) [float] Heat transfer coefficient [W/m^2/K]

Notes

[1] has been reviewed.

References

[1], [2], [3]

Examples

```python
>>> Yun_Heo_Kim(m=1, x=0.4, D=0.3, rhol=567., mul=156E-6, sigma=0.02, Hvap=9E5,
˓→q=1E4)
9479.313988550184
```
2.3 Nucleic boiling and critical heat flux (ht.boiling_nucleic)

ht.boiling_nucleic.Rohsenow (rhol, rhog, mul, kl, Cpl, Hvap, sigma, Te=None, q=None, Csf=0.013, n=1.7)

Calculates heat transfer coefficient for a evaporator operating in the nucleate boiling regime according to [2] as presented in [1].

Either heat flux or excess temperature is required.

With Te specified:

\[ h = \mu_L \Delta H_{vap} \left[ \frac{g(\rho_L - \rho_v)}{\sigma} \right]^{0.5} \left[ \frac{C_{p,L} \Delta T_e^{2/3}}{C_{sf} \Delta H_{vap} P r^n_L} \right]^{3/2} \]

With q specified:

\[ h = \left( \mu_L \Delta H_{vap} \left[ \frac{g(\rho_L - \rho_v)}{\sigma} \right]^{0.5} \left[ \frac{C_{p,L} \Delta T_e^{2/3}}{C_{sf} \Delta H_{vap} P r^n_L} \right]^{3/2} q^{2/3} \right)^{1/3} \]

Parameters

- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the produced gas [kg/m^3]
- **mul** [float] Viscosity of liquid [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Cpl** [float] Heat capacity of liquid [J/kg/K]
- **Hvap** [float] Heat of vaporization of the fluid at P, [J/kg]
- **sigma** [float] Surface tension of liquid [N/m]
- **Te** [float, optional] Excess wall temperature, [K]
- **q** [float, optional] Heat flux, [W/m^2]
- **Csf** [float] Rohsenow coefficient specific to fluid and metal [-]
- **n** [float] Constant, 1 for water, 1.7 (default) for other fluids usually [-]

Returns

- **h** [float] Heat transfer coefficient [W/m^2/K]

Notes

No further work is required on this correlation. Multiple sources confirm its form and rearrangement.

References

[1], [2]
Examples

h for water at atmospheric pressure on oxidized aluminum.

```python
>>> Rohsenow(rhol=957.854, rhog=0.595593, kl=0.680, Cpl=4217, ...
Hvap=2.257E6, sigma=0.0589, Te=4.9, Cs0=0.011, n=1.26)
3723.655267067467
```

`ht.boiling_nucleic.McNelly(rhol, rhog, kl, Cpl, Hvap, sigma, P, Te=None, q=None)`

Calculates heat transfer coefficient for a evaporator operating in the nucleate boiling regime according to [2] as presented in [1].

Either heat flux or excess temperature is required.

With `Te` specified:

\[
h = 0.225 \left( \frac{\Delta T_e}{C_{p,l} \cdot H_{vap}} \right)^{0.69} \left( \frac{P_{kL}}{\rho_L - \rho_V} \right)^{0.31} \left( \frac{\rho_L - 1}{\sigma} \right)^{0.33} \right)^{1/0.31}
\]

With `q` specified:

\[
h = 0.225 \left( \frac{q}{C_{p,l} \cdot H_{vap}} \right)^{0.69} \left( \frac{P_{kL}}{\sigma} \right)^{0.31} \left( \frac{\rho_L - 1}{\rho_V} \right)^{0.33}
\]

Parameters

- `rhol` [float] Density of the liquid [kg/m^3]
- `rhog` [float] Density of the produced gas [kg/m^3]
- `kl` [float] Thermal conductivity of liquid [W/m/K]
- `Cpl` [float] Heat capacity of liquid [J/kg/K]
- `Hvap` [float] Heat of vaporization of the fluid at P, [J/kg]
- `sigma` [float] Surface tension of liquid [N/m]
- `P` [float] Saturation pressure of fluid, [Pa]
- `Te` [float, optional] Excess wall temperature, [K]
- `q` [float, optional] Heat flux, [W/m^2]

Returns

- `h` [float] Heat transfer coefficient [W/m^2/K]

Notes

Further examples for this function are desired.

References

[1], [2]
Examples

Water boiling, with excess temperature of 4.3 K.

```python
>>> McNeil(Te=4.3, P=101325, Cpl=4180., kl=0.688, sigma=0.0588,
... Hvap=2.25E6, rhol=958., rhog=0.597)
533.8056972951352
```

ht.boiling_nucleic.Forster_Zuber(rhol, rhog, mul, kl, Cpl, Hvap, sigma, dPsat, Te=None, q=None)

Calculates heat transfer coefficient for an evaporator operating in the nucleate boiling regime according to [2] as presented in [1].

Either heat flux or excess temperature is required.

With Te specified:

\[
h = 0.00122 \left( \frac{k_L^{0.79} C_{p,l}^{0.45} \rho_L^{0.49}}{\sigma^{0.5} \mu_L^{0.29} H_{vap}^{0.24} \rho_V^{0.24}} \right) \Delta T_e^{0.24} \Delta P_{sat}^{0.75}
\]

With q specified:

\[
h = \left[ 0.00122 \left( \frac{k_L^{0.79} C_{p,l}^{0.45} \rho_L^{0.49}}{\sigma^{0.5} \mu_L^{0.29} H_{vap}^{0.24} \rho_V^{0.24}} \right) \Delta P_{sat}^{0.75} q^{0.24} \right]^{-1/2}
\]

**Parameters**

- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the produced gas [kg/m^3]
- **mul** [float] Viscosity of liquid [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Cpl** [float] Heat capacity of liquid [J/kg/K]
- **Hvap** [float] Heat of vaporization of the fluid at P, [J/kg]
- **sigma** [float] Surface tension of liquid [N/m]
- **dPsat** [float] Difference in saturation pressure of the fluid at Te and T, [Pa]
- **Te** [float, optional] Excess wall temperature, [K]
- **q** [float, optional] Heat flux, [W/m^2]

**Returns**

- **h** [float] Heat transfer coefficient [W/m^2/K]

**Notes**

Examples have been found in [1] and [3] and match exactly.

**References**

[1], [2], [3]
Examples

Water boiling, with excess temperature of 4.3K from [1].

```python
forster_zuber(Te=4.3, dPsat=3906*4.3, Cpl=4180., kl=0.688,
... mul=0.275E-3, sigma=0.0588, H vap=2.25E6, rhol=958., rhog=0.597)
3519.9239897462644
```

ht.boiling_nucleic.Montinsky(P, Pc, Te=None, q=None)

Calculates heat transfer coefficient for a evaporator operating in the nucleate boiling regime according to [2] as presented in [1].

Either heat flux or excess temperature is required.

With $Te$ specified:

$$h = (0.00417 P_c^{0.69} \Delta T e^{0.7} [1.8(P/P_c)^{0.17} + 4(P/P_c)^{1.2} + 10(P/P_c)^{10}])^{1/0.3}$$

With $q$ specified:

$$h = 0.00417 P_c^{0.69} q^{0.7} [1.8(P/P_c)^{0.17} + 4(P/P_c)^{1.2} + 10(P/P_c)^{10}]$$

Parameters

- **P** [float] Saturation pressure of fluid, [Pa]
- **Pc** [float] Critical pressure of fluid, [Pa]
- **Te** [float, optional] Excess wall temperature, [K]
- **q** [float, optional] Heat flux, [W/m^2]

Returns

- **h** [float] Heat transfer coefficient [W/m^2/K]

Notes

Formulas has been found consistent in all cited sources. Examples have been found in [1] and [3].

The equation for this function is sometimes given with a constant of 3.7E-5 instead of 0.00417 if critical pressure is not internally converted to kPa. [3] lists a constant of 3.596E-5.

References

[1], [2], [3], [4]

Examples

Water boiling at 1 atm, with excess temperature of 4.3K from [1].

```python
montinsky(P=101325, Pc=22048321, Te=4.3)
```

2.3. Nucleic boiling and critical heat flux (ht.boiling_nucleic)
ht.boiling_nucleic.StephAn Abdelsalam(rhol, rhog, mul, kl, Cpl, Hvap, sigma, Tsat, Te=None, q=None, kw=401.0, rhow=8.96, Cpw=384.0, angle=None, correlation='general')

Calculates heat transfer coefficient for a evaporator operating in the nucleate boiling regime according to [2] as presented in [1]. Five variants are possible.

Either heat flux or excess temperature is required. The forms for Te are not shown here, but are similar to those of the other functions.

\[
h = 0.23 X_1^{0.674} X_2^{0.35} X_3^{0.371} X_5^{0.297} X_8^{-1.73} k_L / d_B
\]

\[
X_1 = \frac{q D_d}{K_L T_{sat}}
\]

\[
X_2 = \frac{\alpha^2 \rho_L}{\sigma D_d}
\]

\[
X_3 = \frac{C_{p,L} T_{sat} D_d^2}{\alpha^2}
\]

\[
X_4 = \frac{H_{vap} D_d^2}{\alpha^2}
\]

\[
X_5 = \frac{D_v}{\rho_L}
\]

\[
X_6 = \frac{C_{p,L} H_L}{k_L}
\]

\[
X_7 = \frac{\rho_W C_{p,W} k_W}{\rho_L C_{p,L} k_L}
\]

\[
X_8 = \frac{\rho_L - \rho V}{\rho_L}
\]

\[
D_b = 0.0146 \theta \sqrt{\frac{2 \sigma}{g(\rho_L - \rho_g)}}
\]

Respectively, the following four correlations are for water, hydrocarbons, cryogenic fluids, and refrigerants.

\[
h = 0.246 \times 10^7 X_1^{0.673} X_2^{-1.58} X_3^{1.26} X_5^{5.22} k_L / d_B
\]

\[
h = 0.0546 X_5^{0.335} X_1^{0.67} X_8^{-4.33} X_4^{0.248} k_L / d_B
\]

\[
h = 4.82 X_1^{0.624} X_7^{0.117} X_3^{0.374} X_4^{-0.329} X_5^{0.257} k_L / d_B
\]

\[
h = 207 X_1^{0.745} X_5^{0.581} X_6^{0.533} k_L / d_B
\]

**Parameters**

- **rhol** [float] Density of the liquid [kg/m³]
- **rhog** [float] Density of the produced gas [kg/m³]
- **mul** [float] Viscosity of liquid [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Cpl** [float] Heat capacity of liquid [J/kg/K]
- **Hvap** [float] Heat of vaporization of the fluid at P, [J/kg]
- **sigma** [float] Surface tension of liquid [N/m]
- **Tsat** [float] Saturation temperature at operating pressure [Pa]
ht.boiling_nucleic.HEDH_Taborek (P, Pc, Te=None, q=None)
Calculates heat transfer coefficient for a evaporator operating in the nucleate boiling regime according to Taborek (1986) as described in [1] and as presented in [2]. Modification of [3].

Either heat flux or excess temperature is required.

With $\text{Te}$ specified:

$$ h = \left( 0.00417 P_c^{0.69} \Delta T e^{0.7} \left[ 2.1 T_r^{0.27} + (9 + (1 - P_r^2)^{-1}) P_r^2 \right] \right)^{1/0.3} $$

With $q$ specified:

$$ h = 0.00417 P_c^{0.69} q^{0.7} \left[ 2.1 T_r^{0.27} + (9 + (1 - P_r^2)^{-1}) P_r^2 \right] $$

**Parameters**

- **P** [float] Saturation pressure of fluid, [Pa]
- **Pc** [float] Critical pressure of fluid, [Pa]
- **Te** [float, optional] Excess wall temperature, [K]
**Parameters**

- **P** [float] Saturation pressure of fluid, [Pa]
- **Pc** [float] Critical pressure of fluid, [Pa]
- **Te** [float, optional] Excess wall temperature, [K]
- **q** [float, optional] Heat flux, [W/m^2]

**Returns**

- **h** [float] Heat transfer coefficient [W/m²/K]

**Notes**

No examples of this are known. Seems to give very different results than other correlations.

**References**

[1]
Examples

Water boiling at 1 atm, with excess temperature of 4.3 K from [1].

```python
>>> Bier(101325., 22048321.0, Te=4.3)
1290.5349471503353
```

ht.boiling_nucleic.Coop\text{er} (P, Pc, MW, Te=None, q=None, Rp=1e-06)

Calculates heat transfer coefficient for a evaporator operating in the nucleate boiling regime according to [2] as presented in [1].

Either heat flux or excess temperature is required.

With $Te$ specified:

$$h = \left( 55\Delta T e^{0.67 \frac{P}{P_c} ^{0.12 - 0.2 \log_{10} R_p}} \right) \left( - \log_{10} \frac{P}{P_c} \right)^{-0.55} \left( MW^{-0.5} \right)^{1/0.33}$$

With $q$ specified:

$$h = 55q e^{0.67 \frac{P}{P_c} ^{0.12 - 0.2 \log_{10} R_p}} \left( - \log_{10} \frac{P}{P_c} \right)^{-0.55} \left( MW^{-0.5} \right)$$

Parameters

- $P$ [float] Saturation pressure of fluid, [Pa]
- $P_c$ [float] Critical pressure of fluid, [Pa]
- $MW$ [float] Molecular weight of fluid, [g/mol]
- $Te$ [float, optional] Excess wall temperature, [K]
- $q$ [float, optional] Heat flux, [W/m^2]
- $Rp$ [float, optional] Roughness parameter of the surface (1 micrometer default) used by Cooper method, [m]

Returns

- $h$ [float] Heat transfer coefficient [W/m^2/K]

Notes

Examples 1 and 2 are for water and benzene, from [1]. Roughness parameter is with an old definition. Accordingly, it is not used by the $h$ function. If unchanged, the roughness parameter’s logarithm gives a value of 0.12 as an exponent of reduced pressure.

References

[1], [2], [3]

Examples

Water boiling at 1 atm, with excess temperature of 4.3 K from [1].
ht.boiling_nucleic.Gorenflo(P, Pc, q=None, Te=None, CASRN=None, h0=None, Ra=4e-07)
Calculates heat transfer coefficient for a pool boiling according to [1] and also presented in [2]. Calculation is
based on the corresponding states law, with a single regression constant per fluid. P and Pc are always required.
Either q or Te may be specified. Either CASRN or h0 may be specified as well. If CASRN is specified and the
fluid is not in the list of those studied, an error is raises.

\[
\frac{h}{h_0} = C_W F(p^*) \left( \frac{q}{q_0} \right)^n
\]
\[
C_W = \left( \frac{R_a}{R_{ao}} \right)^{0.133}
\]
\[
q_0 = 20\,000\,\text{W/m}^2\]
\[
R_{ao} = 0.4\mu\text{m}
\]

For fluids other than water:

\[
n = 0.9 - 0.3p^{*0.3}
\]
\[
f(p^*) = 1.2p^{*0.27} + \left( 2.5 + \frac{1}{1 - p^*} \right) p^*
\]

For water:

\[
n = 0.9 - 0.3p^{*0.15}
\]
\[
f(p^*) = 1.73p^{*0.27} + \left( 6.1 + \frac{0.68}{1 - p^*} \right) p^2
\]

Parameters

- **P** [float] Saturation pressure of fluid, [Pa]
- **Pc** [float] Critical pressure of fluid, [Pa]
- **q** [float, optional] Heat flux, [W/m²]
- **Te** [float, optional] Excess wall temperature, [K]
- **CASRN** [str, optional] CASRN of fluid
- **h0** [float] Reference heat transfer coefficient for Gorenflo method, [W/m²/K]
- **Ra** [float, optional] Roughness parameter of the surface (0.4 micrometer default) for Gorenflo
  method, [m]

Returns

- **h** [float] Heat transfer coefficient [W/m²/K]

Notes

A more recent set of reference heat fluxes is available. Where a range of values was listed for reference heat
fluxes in [1], values from the second edition of [1] were used instead. 44 values are available, all listed in the
dictionary *h0_Gorenflow_1993*. Values range from 2000 to 24000 W/m²/K.
References
[1], [2]

Examples
Water boiling at 3 bar and a heat flux of 2E4 W/m^2/K.

```python
>>> Gorenflo(3E5, 22048320., q=2E4, CASRN='7732-18-5')
3043.344595525422
```

This function handles the calculation of nucleate boiling heat flux and chooses the best method for performing the calculation based on the provided information.

One of Te and q are always required.

**Parameters**
- **Te** [float, optional] Excess wall temperature, [K]
- **q** [float, optional] Heat flux, [W/m^2]
- **Tsat** [float, optional] Saturation temperature at operating pressure [Pa]
- **P** [float, optional] Saturation pressure of fluid, [Pa]
- **dPsat** [float, optional] Difference in saturation pressure of the fluid at Te and T, [Pa]
- **Cpl** [float, optional] Heat capacity of liquid [J/kg/K]
- **kl** [float, optional] Thermal conductivity of liquid [W/m/K]
- **mul** [float, optional] Viscosity of liquid [Pa*s]
- **rhol** [float, optional] Density of the liquid [kg/m^3]
- **sigma** [float, optional] Surface tension of liquid [N/m]
- **Hvap** [float, optional] Heat of vaporization of the fluid at P, [J/kg]
- **rhog** [float, optional] Density of the produced gas [kg/m^3]
- **MW** [float, optional] Molecular weight of fluid, [g/mol]
- **Pc** [float, optional] Critical pressure of fluid, [Pa]
- **Csf** [float, optional] Rohsenow coefficient specific to fluid and metal [-]
- **n** [float, optional] Rohsenow constant, 1 for water, 1.7 (default) for other fluids usually [-]
- **kw** [float, optional] Thermal conductivity of wall (only for cryogenics) [W/m/K]
- **rhow** [float, optional] Density of the wall (only for cryogenics) [kg/m^3]
- **Cpw** [float, optional] Heat capacity of wall (only for cryogenics) [J/kg/K]
- **angle** [float, optional] Contact angle of bubble with wall [degrees]
- **Rp** [float, optional] Roughness parameter of the surface (1 micrometer default) used by *Cooper* method, [m]

2.3. Nucleic boiling and critical heat flux (ht.boiling_nucleic)
**Ra** [float, optional] Roughness parameter of the surface (0.4 micrometer default) for Gorenflo method, [m]

**h0** [float] Reference heat transfer coefficient for Gorenflo method, [W/m\(^2\)/K]

**CAS** [str, optional] CAS of fluid

**Returns**

**h** [float] Nucleate boiling heat flux [W/m\(^2\)]

**Other Parameters**


**Notes**

The methods Stephan-Abdelsalam, Cooper, and Gorenflo all take other arguments as well such as surface roughness or the thermal properties of the wall material. See them for their documentation. These parameters can also be passed as keyword arguments.

```python
>>> h_nucleic(P=3E5, Pc=22048320., q=2E4, CAS='7732-18-5', Ra=1E-6)
3437.7726419934147
```

**Examples**

Water boiling at 3 bar and a heat flux of 2E4 W/m\(^2\)/K.

```python
>>> h_nucleic(P=3E5, Pc=22048320., q=2E4, CAS='7732-18-5')
3043.344595525422
```

Water, known excess temperature of 4.9 K, Rohsenow method

```python
>>> h_nucleic(rhol=957.854, rhog=0.595593, mul=2.79E-4, kl=0.680, Cpl=4217, ...
Hvap=2.257E6, sigma=0.0589, Te=4.9, Csf=0.011, n=1.26,
... Method='Rohsenow')
3723.655267067467
```

**ht.boiling_nucleic.h_nucleic_methods**

This function returns the names of correlations for nucleate boiling heat flux.

**Parameters**

**Te** [float, optional] Excess wall temperature, [K]

**Tsat** [float, optional] Saturation temperature at operating pressure [Pa]

**P** [float, optional] Saturation pressure of fluid, [Pa]

**dPsat** [float, optional] Difference in saturation pressure of the fluid at Te and T, [Pa]

**Cpl** [float, optional] Heat capacity of liquid [J/kg/K]

**kl** [float, optional] Thermal conductivity of liquid [W/m/K]
mul [float, optional] Viscosity of liquid [Pa*s]

rhol [float, optional] Density of the liquid [kg/m^3]

sigma [float, optional] Surface tension of liquid [N/m]

Hvap [float, optional] Heat of vaporization of the fluid at P, [J/kg]

rhog [float, optional] Density of the produced gas [kg/m^3]

MW [float, optional] Molecular weight of fluid, [g/mol]

Pc [float, optional] Critical pressure of fluid, [Pa]

CAS [str, optional] CAS of fluid

check_ranges [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

methods [list[str]] List of methods which can be used to calculate \( h \) with the given inputs

Examples

```python
>>> h_nucleic_methods(P=3E5, Pc=22048320., Te=4.0, CAS='7732-18-5')
['Gorenflo (1993)', 'HEDH-Taborek', 'Bier', 'Montinsky']
```

ht.boiling_nucleic.Zuber \( (\text{sigma}, \text{Hvap}, \text{rhol}, \text{rhog}, K=0.18) \)

Calculates critical heat flux for nucleic boiling of a flat plate or other shape as presented in various sources. \( K = \pi/24 \) is believed to be the original [1] value for \( K \), but 0.149 is now more widely used, a value claimed to be from [2] according to [5]. Cao [4] lists a value of 0.18 for \( K \). The Wolverine Tube data book also lists a value of 0.18, and so it is the default.

\[
q_c = 0.149 H_vap \rho_g^{0.5} \left[ \sigma g (\rho_L - \rho_g) \right]^{0.25}
\]

Parameters

sigma [float] Surface tension of liquid [N/m]

Hvap [float] Heat of vaporization of the fluid at P, [J/kg]

rhol [float] Density of the liquid [kg/m^3]

rhog [float] Density of the produced gas [kg/m^3]

K [float] Constant []

Returns

q: float Critical heat flux [W/m^2]

Notes

No further work is required on this correlation. Multiple sources confirm its form.

References

[1], [2], [3], [4], [5]
Examples

Example from [3]

```python
>>> Zuber(sigma=8.2E-3, Hvap=272E3, rhol=567, rhog=18.09, K=0.149)
444307.22304342285
>>> Zuber(sigma=8.2E-3, Hvap=272E3, rhol=567, rhog=18.09, K=0.18)
536746.9808578263
```

ht.boiling_nucleic.Serth_HEDH(D, sigma, Hvap, rhol, rhog)

Calculates critical heat flux for nucleic boiling of a tube bundle according to [2], citing [3], and using [1] as the original form.

\[ q_c = KH_{vap}\rho_0^{0.5} [\sigma g(\rho_L - \rho_g)]^{0.25} \]

\[ K = 0.123(R^*)^{-0.25} \text{ for } 0.12 < R^* < 1.17 \]

\[ K = 0.118 \]

\[ R^* = \frac{D}{\sigma} \left[ \frac{g(\rho_L - \rho_g)}{\sigma} \right]^{0.5} \]

Parameters

- `D` [float] Diameter of tubes [m]
- `sigma` [float] Surface tension of liquid [N/m]
- `Hvap` [float] Heat of vaporization of the fluid at T, [J/kg]
- `rhol` [float] Density of the liquid [kg/m^3]
- `rhog` [float] Density of the produced gas [kg/m^3]

Returns

- `q`: float Critical heat flux [W/m^2]

Notes

A further source for this would be nice.

References

[1], [2], [3]

Examples

```python
>>> Serth_HEDH(D=0.0127, sigma=8.2E-3, Hvap=272E3, rhol=567, rhog=18.09)
351867.46522901946
```

ht.boiling_nucleic.HEDH_Montinsky(P, Pc)

Calculates critical heat flux in the nucleate boiling regime according to [3] as presented in [1], using an expression modified from [2].

\[ q_c = 367P_c P_r^{0.35} (1 - P_r)^{0.9} \]
Parameters

- **P** [float] Saturation pressure of fluid, [Pa]
- **Pc** [float] Critical pressure of fluid, [Pa]

Returns

- **q** [float] Critical heat flux [W/m^2]

Notes

No further work is required. Units of Pc are kPa internally.

References

[1], [2], [3]

Examples

Example is from [3] and matches to within the error of the algebraic manipulation rounding.

```python
>>> HEDH_Montinsky(P=310.3E3, Pc=2550E3)
398405.66545181436
```

ht.boiling_nucleic.qmax_boiling(rhol=None, rhog=None, sigma=None, Hvap=None, D=None, P=None, Pc=None, Method=None)

This function handles the calculation of nucleate boiling critical heat flux and chooses the best method for performing the calculation.

Preferred methods are ‘Serth-HEDH’ when a tube diameter is specified, and ‘Zuber’ otherwise.

Parameters

- **rhol** [float, optional] Density of the liquid [kg/m^3]
- **rhog** [float, optional] Density of the produced gas [kg/m^3]
- **sigma** [float, optional] Surface tension of liquid [N/m]
- **Hvap** [float, optional] Heat of vaporization of the fluid at T, [J/kg]
- **D** [float, optional] Diameter of tubes [m]
- **P** [float, optional] Saturation pressure of fluid, [Pa]
- **Pc** [float, optional] Critical pressure of fluid, [Pa]

Returns

- **q** [float] Nucleate boiling critical heat flux [W/m^2]

Other Parameters

- **Method** [string, optional] A string of the function name to use; one of (‘Serth-HEDH’, ‘Zuber’, or ‘HEDH-Montinsky’)

2.3. Nucleic boiling and critical heat flux (ht.boiling_nucleic) 43
Examples

```python
>>> qmax_boiling(D=0.0127, sigma=8.2E-3, Hvap=272E3, rhol=567, rhog=18.09)
351867.46522901946
```

ht.boiling_nucleic.qmax_boiling_methods(rhol=None, rhog=None, sigma=None, Hvap=None, D=None, P=None, Pc=None, check_ranges=False)

This function returns a list of methods names which can be used to calculate nucleate boiling critical heat flux. Preferred methods are ‘Serth-HEDH’ when a tube diameter is specified, and ‘Zuber’ otherwise.

Parameters

- `rhol` [float, optional] Density of the liquid [kg/m^3]
- `rhog` [float, optional] Density of the produced gas [kg/m^3]
- `sigma` [float, optional] Surface tension of liquid [N/m]
- `Hvap` [float, optional] Heat of vaporization of the fluid at T, [J/kg]
- `D` [float, optional] Diameter of tubes [m]
- `P` [float, optional] Saturation pressure of fluid, [Pa]
- `Pc` [float, optional] Critical pressure of fluid, [Pa]
- `check_ranges` [bool, optional] Added for Future use only

Returns

- `methods` [list] List of methods which can be used to calculate qmax with the given inputs

Examples

```python
>>> qmax_boiling_methods(D=0.0127, sigma=8.2E-3, Hvap=272E3, rhol=567, rhog=18.09)
['Serth-HEDH', 'Zuber']
```

2.4 Boiling in plate and frame exchangers (ht.boiling_plate)

ht.boiling_plate.h_boiling_Amalfi(m, x, Dh, rhol, rhog, mul, mug, kl, Hvap, sigma, q, A_channel_flow, chevron_angle=45.0)

Calculates the two-phase boiling heat transfer coefficient of a liquid and gas flowing inside a plate and frame heat exchanger, as developed in [1] from a wide range of existing correlations and data sets. Expected to be the most accurate correlation currently available.

For Bond number < 4 (tiny channel case):

$$h = 982 \left( \frac{k_l}{D_h} \right) \left( \frac{\beta}{\beta_{max}} \right)^{1.101} \left( \frac{G^2 D_h}{\rho_m \sigma} \right)^{0.315} \left( \frac{\rho_l}{\rho_g} \right)^{-0.224} \text{Bo}^{0.320}$$

For Bond number >= 4:

$$h = 18.495 \left( \frac{k_l}{D_h} \right) \left( \frac{\beta}{\beta_{max}} \right)^{0.248} (Re_g)^{0.135} (Re_{lo})^{0.351} \left( \frac{\rho_l}{\rho_g} \right)^{-0.223} \text{Bo}^{0.235} \text{Bo}^{0.198}$$

In the above equations, beta max is 45 degrees; Bo is Boiling number; and Bd is Bond number.

Note that this model depends on the specific heat flux involved.
Parameters

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific point in the plate exchanger
- **Dh** [float] Hydraulic diameter of the plate, \( D_h = \frac{4A}{P} \) [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **mul** [float] Viscosity of the liquid [Pa*s]
- **mug** [float] Viscosity of the gas [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Hvap** [float] Heat of vaporization of the fluid at the system pressure, [J/kg]
- **sigma** [float] Surface tension of liquid [N/m]
- **q** [float] Heat flux, [W/m^2]
- **A_channel_flow** [float] The flow area for the fluid, calculated as \( A_{ch} = 2 \cdot \text{width} \cdot \text{amplitude} \) [m]
- **chevron_angle** [float, optional] Angle of the plate corrugations with respect to the vertical axis (the direction of flow if the plates were straight), between 0 and 90. For exchangers with two angles, use the average value. [degrees]

Returns

- **h** [float] Boiling heat transfer coefficient [W/m^2/K]

Notes

Heat transfer correlation developed from 1903 datum. Fluids included R134a, ammonia, R236fa, R600a, R290, R1270, R1234yf, R410A, R507A, ammonia/water, and air/water mixtures. Wide range of operating conditions, plate geometries.

References

[1]

Examples

```python
>>> h_boiling_Amalfi(m=3E-5, x=.4, Dh=0.00172, rhol=567., rhog=18.09,
... kl=0.086, mul=156E-6, mug=7.11E-6, sigma=0.02, Hvap=9E5, q=1E5,
... A_channel_flow=0.0003)
776.0781179096225
```

Calculates the two-phase boiling heat transfer coefficient of a liquid and gas flowing inside a plate and frame heat exchanger, as shown in [1] and reviewed in [2].

For \( Re_g/Re_l < 9 \):

\[
h = 98.7 \left( \frac{k_l}{D_h} \right) \left( \frac{Re_g}{Re_l} \right)^{-0.0848} \cdot Bo^{-0.0597} \cdot X_{H}^{0.0973}
\]
For $\frac{Re_g}{Re_l} \geq 9$:

\[
h = 234.9 \left( \frac{k_l}{D_h} \right) \left( \frac{Re_g}{Re_l} \right)^{-0.576} Bo^{-0.275} X_{tt}^{0.66}
\]

\[
X_{tt} = \left( \frac{1 - x}{x} \right)^{0.875} \left( \frac{\rho_g}{\rho_l} \right)^{0.5} \left( \frac{\mu_l}{\mu_g} \right)^{0.125}
\]

In the above equations, $Bo$ is Boiling number.

Note that this model depends on the specific heat flux involved. It also uses equivalent diameter, not hydraulic diameter.

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific point in the plate exchanger []
- **D_eq** [float] Equivalent diameter of the channels, $D_{eq} = 4a$ [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **mul** [float] Viscosity of the liquid [Pa*s]
- **mug** [float] Viscosity of the gas [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Hvap** [float] Heat of vaporization of the fluid at the system pressure, [J/kg]
- **q** [float] Heat flux, [W/m^2]
- **A_channel_flow** [float] The flow area for the fluid, calculated as $A_{ch} = 2 \cdot \text{width} \cdot \text{amplitude}$ [m]

**Returns**

- **h** [float] Boiling heat transfer coefficient [W/m^2/K]

**Notes**

This correlation was developed with mass fluxes from 14.5 to 33.6 kg/m^2/s, heat flux from 15 to 30 kW/m^2, qualities from 0.09 to 0.6, $200 < Re < 600$, $2.3 < \frac{Re_g}{Re_l} < 32.1$, $0.00019 < Bo < 0.001$, $0.028 < X_{tt} < 0.3$. Mean average deviation of 4.4%.

**References**

[1], [2]

**Examples**

```python
>>> h_boiling_Lee_Kang_Kim(m=3E-5, x=.4, D_eq=0.002, rhol=567., rhog=18.09, ...
... kl=0.086, mul=156E-6, mug=9E-6, Hvap=9E5, q=1E5, A_channel_flow=0.0003)
1229.6271295086806
```
Calculates the two-phase boiling heat transfer coefficient of a liquid and gas flowing inside a plate and frame heat exchanger, as developed in [1] from experiments with three plate exchangers and the working fluids R410A and R22. A well-documented and tested correlation, reviewed in [2], [3], [4], [5], and [6].

\[
  h = G_e 1 \left( \frac{k_l}{D_h} \right) R_e^{G_e 2} P_r^{0.4} B_o^{0.3}
\]

\[
  G_e 1 = 2.81 \left( \frac{\lambda}{D_h} \right)^{-0.041} \left( \frac{\pi}{2} - \beta \right)^{-2.83}
\]

\[
  G_e 2 = 0.746 \left( \frac{\lambda}{D_h} \right)^{-0.082} \left( \frac{\pi}{2} - \beta \right)^{0.61}
\]

\[
  R_e = \frac{G_e q D_h}{\mu_l}
\]

\[
  B_o = \frac{q}{G_e q H_vap}
\]

\[
  G_e q = \frac{m}{A_{flow}} \left[ 1 - x + x \left( \frac{\rho_l}{\rho_g} \right)^{1/2} \right]
\]

In the above equations, lambda is the wavelength of the corrugations, and the flow area is specified to be (twice the corrugation amplitude times the width of the plate. The mass flow is that per channel. Radians is used in degrees, and the formulas are for the inclination angle not the chevron angle (it is converted internally). Note that this model depends on the specific heat flux involved.

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific point in the plate exchanger []
- **Dh** [float] Hydraulic diameter of the plate, \( D_h = \frac{4A}{\phi} \) [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **mul** [float] Viscosity of the liquid [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Hvap** [float] Heat of vaporization of the fluid at the system pressure, [J/kg]
- **Cpl** [float] Heat capacity of liquid [J/kg/K]
- **q** [float] Heat flux, [W/m^2]
- **A_channel_flow** [float] The flow area for the fluid, calculated as \( A_{ch} = 2 \cdot \text{width} \cdot \text{amplitude} \) [m]
- **wavelength** [float] Distance between the bottoms of two of the ridges (sometimes called pitch), [m]
- **chevron_angle** [float, optional] Angle of the plate corrugations with respect to the vertical axis (the direction of flow if the plates were straight), between 0 and 90. For exchangers with two angles, use the average value. [degrees]

**Returns**

- **h** [float] Boiling heat transfer coefficient [W/m^2/K]

---

2.4. Boiling in plate and frame exchangers (ht.boiling_plate)
Notes

Date regression was with the log mean temperature difference, uncorrected for geometry. Developed with three plate heat exchangers with angles of 45, 35, and 20 degrees. Mass fluxes ranged from 13 to 34 kg/m^2/s; evaporating temperatures of 5, 10, and 15 degrees, vapor quality 0.9 to 0.15, heat fluxes of 2.5-8.5 kW/m^2.

References

[1], [2], [3], [4], [5], [6]

Examples

```python
>>> h_boiling_Han_Lee_Kim(m=3E-5, x=.4, Dh=0.002, rhol=567., rhog=18.09,
... kl=0.086, mul=156E-6, Hvap=9E5, Cpl=2200, q=1E5, A_channel_flow=0.0003,
... wavelength=3.7E-3, chevron_angle=45)
675.73225419421
```

ht.boiling_plate.h_boiling_Huang_Sheer(rhol, rhog, mul, kl, Hvap, sigma, Cpl, q, Tsat, angle=35.0)

Calculates the two-phase boiling heat transfer coefficient of a liquid and gas flowing inside a plate and frame heat exchanger, as developed in [1] and again in the thesis [2]. Depends on the properties of the fluid and not the heat exchanger’s geometry.

\[
h = 1.87 \times 10^{-3} \left( \frac{k_l}{d_o} \right) \left( \frac{q d_o}{k_l T_{sat}} \right)^{0.56} \left( \frac{H_{vap} d_o^2}{\alpha_l^2} \right)^{0.31} \frac{P_f l^{0.33}}{T_{sat}}
\]

\[
d_o = 0.0146 \theta \left[ \frac{2\sigma}{g(\rho_l - \rho_g)} \right]^{0.5} \theta = 35^\circ
\]

Note that this model depends on the specific heat flux involved and the saturation temperature of the fluid.

Parameters

rhol [float] Density of the liquid [kg/m^3]
rhog [float] Density of the gas [kg/m^3]
mul [float] Viscosity of the liquid [Pa*s]
kl [float] Thermal conductivity of liquid [W/m/K]
Hvap [float] Heat of vaporization of the fluid at the system pressure, [J/kg]
sigma [float] Surface tension of liquid [N/m]
Cpl [float] Heat capacity of liquid [J/kg/K]
q [float] Heat flux, [W/m^2]
Tsat [float] Actual saturation temperature of the fluid at the system pressure, [K]
angle [float, optional] Contact angle of the bubbles with the wall, assumed 35 for refrigerants in the development of the correlation [degrees]

Returns

h [float] Boiling heat transfer coefficient [W/m^2/K]
Notes

Developed with 222 data points for R134a and R507A with only two of them for ammonia and R12. Chevron angles ranged from 28 to 60 degrees, heat fluxes from 1.85 kW/m² to 10.75 kW/m², mass fluxes 5.6 to 52.25 kg/m²/s, qualities from 0.21 to 0.95, and saturation temperatures in degrees Celsius of 1.9 to 13.04.

The inclusion of the saturation temperature makes this correlation have limited predictive power for other fluids whose saturation temperatures might be much higher or lower than those used in the development of the correlation. For this reason it should be regarded with caution.

As first published in [1] a power of two was missing in the correlation for bubble diameter in the dimensionless group with a power of 0.31. That made the correlation non-dimensional.

A second variant of this correlation was also published in [2] but with less accuracy because it was designed to mimic the standard pool boiling curve.

The correlation is reviewed in [3], but without the corrected power. It was also changed there to use hydraulic diameter, not bubble diameter. It still ranked as one of the more accurate correlations reviewed. [4] also reviewed it without the corrected power but found it predicted the lowest results of those surveyed.

References

[1], [2], [3], [4]

Examples

```python
>>> h_boiling_Huang_Sheer(rhol=567., rhog=18.09, kl=0.086, mul=156E-6,
... Hvap=9E5, sigma=0.02, Cpl=2200, q=1E4, Tsat=279.15)
4401.055635078054

ht.boiling_plate.h_boiling_Yan_Lin(m, x, Dh, rhol, rhog, mul, kl, Hvap, Cpl, q, A_channel_flow)
Calculates the two-phase boiling heat transfer coefficient of a liquid and gas flowing inside a plate and frame heat exchanger, as developed in [1]. Reviewed in [2], [3], [4], and [5].

\[
h = 1.926 \left( \frac{k_l}{D_h} \right) Re_{eq} Pr_1^{1/3} Bo_{eq}^{0.3} Re^{-0.5}
\]

\[
Re_{eq} = \frac{Geq D_h}{\mu_l}
\]

\[
Bo_{eq} = \frac{q}{Geq Hvap}
\]

\[
Geq = \frac{m}{A_{flow}} \left[ 1 - x + x \left( \frac{\mu_l}{\nu_g} \right)^{1/2} \right]
\]

\[
Re = \frac{GD_h}{\mu_l}
\]

Claimed to be valid for 2000 < Re_{eq} < 10000.

Parameters

- `m` [float] Mass flow rate [kg/s]
- `x` [float] Quality at the specific point in the plate exchanger []
- `Dh` [float] Hydraulic diameter of the plate, $D_h = \frac{4A}{\pi}$ [m]

2.4. Boiling in plate and frame exchangers (ht.boiling_plate)
**rhol** [float] Density of the liquid [kg/m^3]

**rhog** [float] Density of the gas [kg/m^3]

**mul** [float] Viscosity of the liquid [Pa*s]

**kl** [float] Thermal conductivity of liquid [W/m/K]

**Hvap** [float] Heat of vaporization of the fluid at the system pressure, [J/kg]

**Cpl** [float] Heat capacity of liquid [J/kg/K]

**q** [float] Heat flux, [W/m^2]

**A_channel_flow** [float] The flow area for the fluid, calculated as $A_{ch} = 2 \cdot \text{width} \cdot \text{amplitude}$ [m]

**Returns**

**h** [float] Boiling heat transfer coefficient [W/m^2/K]

**Notes**

Developed with R134a as the refrigerant in a PHD with 2 channels, chevron angle 60 degrees, quality from 0.1 to 0.8, heat flux 11-15 kW/m^2, and mass fluxes of 55 and 70 kg/m^2/s.

**References**

[1], [2], [3], [4], [5]

**Examples**

```python
>>> h_boiling_Yan_Lin(m=3E-5, x=.4, Dh=0.002, rhol=567., rhog=18.09,
... kl=0.086, Cpl=2200, mul=156E-6, Hvap=9E5, q=1E5, A_channel_flow=0.0003)
318.7228565961241
```

## 2.5 Condensation (ht.condensation)

**ht.condensation.Boyko_Kruzhilin** *(m, rhog, rhol, kl, mul, Cpl, D, x)*

Calculates heat transfer coefficient for condensation of a pure chemical inside a vertical tube or tube bundle, as presented in [2] according to [1].

\[
h_f = h_{LO} \left[ 1 + x \left( \frac{\rho_L}{\rho_G} - 1 \right) \right]^{0.5}
\]

\[
h_{LO} = 0.021 \frac{k_L}{L} \text{Re}_{LO}^{0.8} Pr^{0.43}
\]

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **rhog** [float] Density of the gas [kg/m^3]
- **rhol** [float] Density of the liquid [kg/m^3]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
**Returns**

- `h` [float] Heat transfer coefficient [W/m²/K]

**Notes**

To calculate overall heat transfer coefficient during condensation, simply average values at \( x = 1 \) and \( x = 0 \).

**References**

[1], [2]

**Examples**

Page 589 in [2], matches exactly.

```python
>>> Boyko_Kruzhilin(m=500*pi/4.*.03**2, rhog=6.36, rhol=582.9, kl=0.098, 
... mul=159E-6, Cpl=2520., D=0.03, x=0.85)
10598.657227479956
```

**ht.condensation.Nusselt_laminar** (*Tsat, Tw, rhog, rhol, kl, mul, Hvap, L, angle=90.0*)

Calculates heat transfer coefficient for laminar film condensation of a pure chemical on a flat plate, as presented in [1] according to an analysis performed by Nusselt in 1916.

\[
h = 0.943 \left[ \frac{g \sin(\theta) \rho_l (\rho_l - \rho_v) k_l^3 \Delta H_{vap}}{\mu_l (T_{sat} - T_w) L} \right]^{0.25}
\]

**Parameters**

- `Tsat` [float] Saturation temperature at operating pressure [Pa]
- `Tw` [float] Wall temperature, [K]
- `rhog` [float] Density of the gas [kg/m³]
- `rhol` [float] Density of the liquid [kg/m³]
- `kl` [float] Thermal conductivity of liquid [W/m/K]
- `mul` [float] Viscosity of liquid [Pa*s]
- `Hvap` [float] Heat of vaporization of the fluid at P, [J/kg]
- `L` [float] Length of the plate [m]
- `angle` [float, optional] Angle of inclination of the plate [degrees]

**Returns**

- `h` [float] Heat transfer coefficient [W/m²/K]
Notes

Optionally, the plate may be inclined. The constant 0.943 is actually:

\[ 2\sqrt{2}/3 \]

References

[1]

Examples

p. 578 in [1], matches exactly.

```python
>>> Nusselt_laminar(Tsat=370, Tw=350, rhog=7.0, rhol=585., kl=0.091,
... mul=158.9E-6, Hvap=776900, L=0.1)
1482.206403453679
```

ht.condensation.h_kinetic(T, P, MW, Hvap, f=1.0)
Calculates heat transfer coefficient for condensation of a pure chemical inside a vertical tube or tube bundle, as presented in [2] according to [1].

\[
h = \left( \frac{2f}{2 - f} \right) \left( \frac{MW}{1000 \cdot 2\pi RT} \right)^{0.5} \left( \frac{H_{vap}^2 P \cdot MW}{1000 \cdot RT^2} \right)
\]

Parameters

- **T**  [float] Vapor temperature, [K]
- **P**  [float] Vapor pressure, [Pa]
- **MW**  [float] Molecular weight of the gas, [g/mol]
- **Hvap**  [float] Heat of vaporization of the fluid at P, [J/kg]
- **f**  [float] Correction factor, [-]

Returns

- **h**  [float] Heat transfer coefficient [W/m^2/K]

Notes

f is a correction factor for how the removal of gas particles affects the behavior of the ideal gas in diffusing to the condensing surface. It is quite close to one, and has not been well explored in the literature due to the rarity of the importance of the kinetic resistance.

References

[1], [2], [3]
Examples

Water at 1 bar and 300 K:

```python
>>> h_kinetic(300, 1E5, 18.02, 2441674)
30788829.908851154
```

ht.condensation.Akers_Deans_Crosser(m, rhog, rhol, kl, mul, Cpl, D, x)
Calculates heat transfer coefficient for condensation of a pure chemical inside a vertical tube or tube bundle, as presented in [2] according to [1].

\[ Nu = \frac{hD_i}{k_l} = C Re_e^n Pr_l^{1/3} \]

\[ C = 0.0265, n = 0.8 \text{ for } Re_e > 5 \times 10^4 \]

\[ C = 5.03, n = \frac{1}{3} \text{ for } Re_e < 5 \times 10^4 \]

\[ Re_e = \frac{D_i G_e}{\mu_l} \]

\[ G_e = G \left[ (1 - x) + x (\rho_l / \rho_g)^{0.5} \right] \]

Parameters

- **m** [float] Mass flow rate [kg/s]
- **rhog** [float] Density of the gas [kg/m^3]
- **rhol** [float] Density of the liquid [kg/m^3]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **mul** [float] Viscosity of liquid [Pa*s]
- **Cpl** [float] Constant-pressure heat capacity of liquid [J/kg/K]
- **D** [float] Diameter of the tubing [m]
- **x** [float] Quality at the specific interval [-]

Returns

- **h** [float] Heat transfer coefficient [W/m^2/K]

References

[1], [2]

Examples

```python
>>> Akers_Deans_Crosser(m=0.35, rhog=6.36, rhol=582.9, kl=0.098, ...
... mul=159E-6, Cpl=2520., D=0.03, x=0.85)
7117.24177265201
```

ht.condensation.Cavallini_Smith_Zecchin(m, x, D, rhol, rhog, mul, mug, kl, Cpl)
Calculates heat transfer coefficient for condensation of a fluid inside a tube, as presented in [1], also given in [2] and [3].

\[ Nu = \frac{hD_i}{k_l} = 0.05 Re_e^{0.8} Pr_l^{0.33} \]
\[ Re_{eq} = Re_g (\mu_g/\mu_l) (\rho_l/\rho_g)^{0.5} + Re_l \]

\[ v_{gs} = \frac{m x}{\rho_g \frac{\pi}{4} D^2} \]

\[ v_{ls} = \frac{m (1 - x)}{\rho_l \frac{\pi}{4} D^2} \]

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific interval [-]
- **D** [float] Diameter of the channel [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **mul** [float] Viscosity of liquid [Pa*s]
- **mug** [float] Viscosity of gas [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **Cpl** [float] Constant-pressure heat capacity of liquid [J/kg/K]

**Returns**

- **h** [float] Heat transfer coefficient [W/m^2/K]

**References**

[1], [2], [3]

**Examples**

```python
cavallini_smith_zecchin(m=1, x=0.4, D=.3, rhol=800, rhog=2.5, mul=1E-5, 
mug=1E-3, kl=0.6, Cpl=2300)
```

ht.condensation.Shah (m, x, D, rhol, mul, kl, Cpl, P, Pc)

Calculates heat transfer coefficient for condensation of a fluid inside a tube, as presented in [1] and again by the same author in [2]; also given in [3]. Requires no properties of the gas. Uses the Dittus-Boelter correlation for single phase heat transfer coefficient, with a Reynolds number assuming all the flow is liquid.

\[ h_{TP} = h_L \left[ (1 - x)^{0.8} + \frac{3.8 x^{0.76} (1 - x)^{0.94}}{P_r^{0.38}} \right] \]

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific interval [-]
- **D** [float] Diameter of the channel [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **mul** [float] Viscosity of liquid [Pa*s]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
\[ C_{pl} \text{ [float] Constant-pressure heat capacity of liquid [J/kg/K]} \]

\[ P \text{ [float] Pressure of the fluid, [Pa]} \]

\[ P_{c} \text{ [float] Critical pressure of the fluid, [Pa]} \]

**Returns**

\[ h \text{ [float] Heat transfer coefficient [W/m^2/K]} \]

**Notes**

[1] is well written an unambiguous as to how to apply this equation.

**References**

[1], [2], [3]

**Examples**

```
>>> Shah(m=1, x=0.4, D=.3, rhol=800, mul=1E-5, kl=0.6, Cpl=2300, P=1E6, P_c=2E7)
2561.2593415479214
```

## 2.6 Conduction and shape factors (ht.conduction)

**ht.conduction.R_to_k**( \( R, t, A=1.0 \))

Returns the thermal conductivity of a substance given its thickness and thermal resistance.

\[
k = \frac{t}{RA}
\]

**Parameters**

- \( R \) [float] Thermal resistance of a substance, (K/W) if \( A \) is 1 m^2, otherwise must be [m^2*K/W]
- \( t \) [float] Thickness of the substance used in the measurement of \( R \), [m]
- \( A \) [float, optional] Area; normally 1, [m^2]

**Returns**

- \( k \) [float] Thermal conductivity of a substance [W/m/K]

**Notes**

When solving problems of changing areas, this value may be calculated with an area other than 1 m^2. Values in tables reported as properties of materials are often divided by area already; the conversion holds if \( A \) is 1.

**References**

[1]
Examples

```python
>>> R_to_k(R=0.05, t=0.025)
0.5
```

ht.conduction.k_to_R(k, t, A=1.0)

Returns the thermal resistance of a substance given its thickness and thermal conductivity.

\[ R = \frac{t}{kA} \]

Parameters

- **k** [float] Thermal conductivity of a substance [W/m/K]
- **t** [float] Thickness of the substance for a given value of R, [m]
- **A** [float, optional] Area; normally 1, [m^2]

Returns

- **R** [float] Thermal resistance of a substance [K/W]

Notes

When solving problems of changing areas, this value may be calculated with an area other than 1 m^2. Values in tables reported as properties of materials are often divided by area already; the conversion holds if A is 1.

References

[1]

Examples

```python
>>> k_to_R(k=0.5, t=0.025)
0.05
```

ht.conduction.k_to_thermal_resistivity(k)

Returns the thermal resistivity of a substance given its thermal conductivity.

\[ r = \frac{1}{k} \]

Parameters

- **k** [float] Thermal conductivity of a substance [W/m/K]

Returns

- **r** [float] Thermal resistivity of a substance [m*K/W]

Notes

Do not confuse this with thermal resistance! Often not introduced in heat transfer textbooks to avoid further confusion. Used almost exclusively as a description of solids. Thermal resistivity has different units than R-value, but is of the same dimensionality.
References

[1]

Examples

```python
>>> k_to_thermal_resistivity(0.25)
4.0
```

ht.conduction.thermal_resistivity_to_k(r)
Returns the thermal resistivity of a substance given its thermal conductivity.

\[ k = \frac{1}{r} \]

Parameters

- r [float] Thermal resistivity of a substance [m*K/W]

Returns

- k [float] Thermal conductivity of a substance [W/m/K]

Notes

Do not confuse this with thermal resistance! Often not introduced in heat as a description of solids. Thermal resistivity has different units than R-value, but is of the same dimensionality.

References

[1]

Examples

```python
>>> thermal_resistivity_to_k(4)
0.25
```

ht.conduction.R_value_to_k(R_value, SI=True)
Returns the thermal conductivity of a substance given its R-value, which can be in either SI units of m² K/(W*inch) or the Imperial units of ft² deg F*h/(BTU*inch).

Parameters

- R_value [float] R-value of a substance [m² K/(W*inch) or ft² deg F*h/(BTU*inch)]
- SI [bool, optional] Whether to use the SI conversion or not

Returns

- k [float] Thermal conductivity of a substance [W/m/K]

Notes

If given input is SI, it is divided by 0.0254 (multiplied by 39.37) and then inversed. Otherwise, it is multiplied by 6.93347 and then inversed.
ht.conduction.k_to_R_value(k, SI=True)

Returns the R-value of a substance given its thermal conductivity. Will return R-value in SI units unless SI is false. SI units are \( m^2 K/(W*inch) \); Imperial units of R-value are \( ft^2 \) deg \( F/h/(BTU*inch) \).

**Parameters**
- **k** [float] Thermal conductivity of a substance [W/m/K]
- **SI** [bool, optional] Whether to use the SI conversion or not

**Returns**
- **R_value** [float] R-value of a substance [\( m^2 K/(W*inch) \) or \( ft^2 \) deg \( F/h/(BTU*inch) \)]

**Notes**
Provides the reverse conversion of 'R_value_to_k'.

**References**
[1]

**Examples**

```python
>>> k_to_R_value(R_value_to_k(0.12)), k_to_R_value(R_value_to_k(0.71, SI=False))
(0.11999999999999998, 0.7099999999999999)
```

ht.conduction.R_cylinder(Di, Do, k, L)

Returns the thermal resistance \( R \) of a cylinder of constant thermal conductivity \( k \), of inner and outer diameter \( Di \) and \( Do \), and with a length \( L \).

\[
(hA)_{cylinder} = \frac{k}{\ln(D_o/D_i)} \cdot \frac{2\pi L}{2\pi Lk} = \frac{1}{(hA)_{cylinder}} = \frac{\ln(D_o/D_i)}{2\pi Lk}
\]

**Parameters**
- **Di** [float] Inner diameter of the cylinder, [m]
- **Do** [float] Outer diameter of the cylinder, [m]
\[ k \] [float] Thermal conductivity of the cylinder, [W/m/K]
\[ L \] [float] Length of the cylinder, [m]

Returns
\[ R \] [float] Thermal resistance [K/W]

References
[1]

Examples

```python
>>> R_cylinder(0.9, 1., 20, 10)
8.38432343682705e-05
```

ht.conduction.S_isothermal_sphere_to_plane(D, Z)
Returns the Shape factor \( S \) of a sphere of constant temperature and of outer diameter \( D \) which is \( Z \) distance from an infinite plane.
\[
S = \frac{2\pi D}{1 - \frac{D}{4Z}}
\]

Parameters
- \( D \) [float] Diameter of the sphere, [m]
- \( Z \) [float] Distance from the middle of the sphere to the infinite plane, [m]

Returns
- \( S \) [float] Shape factor [m]

Notes
No restrictions on the use of this equation.
\[
Q = Sk(T_1 - T_2)
\]
\[
R_{shape} = \frac{1}{Sk}
\]

References
[1], [2]

Examples

```python
>>> S_isothermal_sphere_to_plane(1, 100)
6.298932638776527
```
**ht.conduction.S_isothermal_pipe_to_plane**(\(D, Z, L=1\))

Returns the Shape factor \(S\) of a pipe of constant outer temperature and of outer diameter \(D\) which is \(Z\) distance from an infinite plane. Length \(L\) must be provided, but can be set to 1 to obtain a dimensionless shape factor used in some sources.

\[
S = \frac{2\pi L}{\cosh^{-1}(2Z/D)}
\]

**Parameters**

- \(D\) [float] Diameter of the pipe, [m]
- \(Z\) [float] Distance from the middle of the pipe to the infinite plane, [m]
- \(L\) [float, optional] Length of the pipe, [m]

**Returns**

- \(S\) [float] Shape factor [m]

**Notes**

\(L\) should be much larger than \(D\).

\[
Q = Sk(T_1 - T_2)
\]

\[
R_{\text{shape}} = \frac{1}{Sk}
\]

**References**

[1], [2]

**Examples**

```python
>>> S_isothermal_pipe_to_plane(1, 100, 3)
3.146071454894645
```

**ht.conduction.S_isothermal_pipe_normal_to_plane**(\(D, L\))

Returns the Shape factor \(S\) of a pipe of constant outer temperature and of outer diameter \(D\) which extends into an infinite medium below an an infinite plane.

\[
S = \frac{2\pi L}{\ln(4L/D)}
\]

**Parameters**

- \(D\) [float] Diameter of the pipe, [m]
- \(L\) [float] Length of the pipe, [m]

**Returns**

- \(S\) [float] Shape factor [m]
Notes

L should be much larger than D.

\[
Q = Sk(T_1 - T_2)
\]

\[
R_{\text{shape}} = \frac{1}{Sk}
\]

References

[1], [2]

Examples

```python
>>> S_isothermal_pipe_normal_to_plane(1, 100)
104.86893910124888
```

ht.conduction.S_isothermal_pipe_to_isothermal_pipe(D1, D2, W, L=1.0)

Returns the Shape factor \(S\) of a pipe of constant outer temperature and of outer diameter \(D1\) which is \(W\) distance from another infinite pipe of outer diameter \(D2\). Length \(L\) must be provided, but can be set to 1 to obtain a dimensionless shape factor used in some sources.

\[
S = \frac{2\pi L}{\cosh^{-1}\left(\frac{4w^2-D_1^2-D_2^2}{2D_1D_2}\right)}
\]

Parameters

- **D1** [float] Diameter of one pipe, [m]
- **D2** [float] Diameter of the other pipe, [m]
- **W** [float] Distance from the middle of one pipe to the middle of the other, [m]
- **L** [float, optional] Length of the pipe, [m]

Returns

- **S** [float] Shape factor [m]

Notes

L should be much larger than both diameters. L should be larger than \(W\).

\[
Q = Sk(T_1 - T_2)
\]

\[
R_{\text{shape}} = \frac{1}{Sk}
\]

References

[1], [2]

Examples
S_isothermal_pipe_to_isothermal_pipe(.1, .2, 1, 1) 1.188711034982268

ht.conduction. S_isothermal_pipe_to_two_planes(D, Z, L=1.0)
Returns the Shape factor S of a pipe of constant outer temperature and of outer diameter D which is Z distance from two infinite isothermal planes of equal temperatures, parallel to each other and enclosing the pipe. Length L must be provided, but can be set to 1 to obtain a dimensionless shape factor used in some sources.

\[
S = \frac{2\pi L}{\ln \frac{8z}{\pi D}}
\]

Parameters
- D [float] Diameter of the pipe, [m]
- Z [float] Distance from the middle of the pipe to either of the planes, [m]
- L [float, optional] Length of the pipe, [m]

Returns
- S [float] Shape factor [m]

Notes
L should be much larger than both diameters. L should be larger than W.

\[
Q = SK(T_1 - T_2)
\]

\[
R_{\text{shape}} = \frac{1}{SK}
\]

References
[1], [2]

Examples

S_isothermal_pipe_to_two_planes(.1, 5, 1) 1.2963749299921428

S_isothermal_pipe_eccentric_to_isothermal_pipe(D1, D2, Z, L=1.0)
Returns the Shape factor S of a pipe of constant outer temperature and of outer diameter D1 which is Z distance from the center of another pipe of outer diameter D2. Length L must be provided, but can be set to 1 to obtain a dimensionless shape factor used in some sources.

\[
S = \frac{2\pi L}{\cosh^{-1} \left( \frac{D_1^2 + D_2^2 - 4Z^2}{2D_1D_2} \right)}
\]

Parameters
- D1 [float] Diameter of inner pipe, [m]
- D2 [float] Diameter of outer pipe, [m]
- Z [float] Distance from the middle of inner pipe to the center of the other, [m]
- L [float, optional] Length of the pipe, [m]
Returns

S [float] Shape factor [m]

Notes

L should be much larger than both diameters. D2 should be larger than D1.

\[
Q = Sk(T_1 - T_2) \\
R_{\text{shape}} = \frac{1}{Sk}
\]

References

[1], [2]

Examples

```python
>>> S_isothermal_pipe_eccentric_to_isothermal_pipe(.1, .4, .05, 10)
47.709841915608976
```

ht.conduction.cylindrical_heat_transfer (Ti, To, hi, ho, Di, ts, ks)

Calculation for the heat transfer through a cylindrical wall, as occurs in pipes and cylindrical vessels. This is the core method which calculates the temperatures of each layer - and allows an outer layer to iterate on temperature or duty to meet a fixed specification, or include things like temperature dependent thermal conductivities or radiation.

Parameters

- Ti [float] Temperature of the inside of the cylinder, [K]
- To [float] External temperature outside the cylinder, away from the cylinder wall, [K]
- hi [float] Inside heat transfer coefficient, [W/m^2/K]
- ho [float] Outside heat transfer coefficient, [W/m^2/K]
- Di [float] Inside diameter of cylinder, [m]
- ts [list[float]] List of thicknesses of each layer of the cylinder, [m]
- ks [list[float]] List of thermal conductivities of each layer of the cylinder, [w/m/K]

Returns

results [dict]

- Q : Heat exchanged through the cylinder (per meter of length), [W/m]
- Rs : Thermal resistances of each of the layers, [m*K/W]
- Ts : Temperatures of the outside of each of the layers, [K]
- UA [Heat transfer coefficient times area (on a per-meter of ) cylinder) basis, [W/K/m]
- U_inner [Heat transfer coefficient with respect to the inside] diameter, [W/K]
- U_outer [Heat transfer coefficient with respect to the exterior] diameter, [W/K]
- q [Specific heat exchanged (per square meter) through the cylinder] (per meter of length), [W/m^3]
Examples

```python
>>> pprint(cylindrical_heat_transfer(Ti=453.15, To=301.15, hi=1e12, ho=22.697193,
    Di=0.0779272, ts=[0.0054864, .05], ks=[56.045, 0.059853265]))
{'Q': 73.12000884069367,
 'Rs': [0.00022201030738405449, 1.189361782070256],
 'Ts': [453.15, 453.1226455779877, 306.578530147744],
 'UA': 0.48105268974140575,
 'U_inner': 1.9649599487726137,
 'U_outer': 0.8106078714663484,
 'q': 123.21239646288495}
```

2.7 External convection (ht.conv_external)

ht.conv_external.Nu_cylinder_Zukauskas(Re, Pr, Prw=None)
Calculates Nusselt number for crossflow across a single tube at a specified Re. Method from [1], also shown without modification in [2]. This method applies to both the laminar and turbulent regimes.

\[
N_u D = C Re^m Pr^n \left( \frac{Pr}{Pr_w} \right)^{1/4}
\]

Parameters

- **Re** [float] Reynolds number with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at free stream temperature [-]
- **Prw** [float, optional] Prandtl number at wall temperature, [-]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

Notes

If Prandtl number at wall are not provided, the Prandtl number correction is not used and left to an outside function.

- n is 0.37 if Pr <= 10; otherwise n is 0.36.
- C and m are from the following table. If Re is outside of the ranges shown, the nearest range is used blindly.

<table>
<thead>
<tr>
<th>Re</th>
<th>C</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-40</td>
<td>0.75</td>
<td>0.4</td>
</tr>
<tr>
<td>40-1E3</td>
<td>0.51</td>
<td>0.5</td>
</tr>
<tr>
<td>1E3-2E5</td>
<td>0.26</td>
<td>0.6</td>
</tr>
<tr>
<td>2E5-1E6</td>
<td>0.076</td>
<td>0.7</td>
</tr>
</tbody>
</table>

References

[1], [2]
Examples

Example 7.3 in [2], matches.

```python
>>> Nu_cylinder_Zukauskas(7992, 0.707, 0.69)
50.523612661934386
```

ht.conv_external.Nu_cylinder_Churchill_Bernstein (Re, Pr)
Calculates Nusselt number for crossflow across a single tube at a specified Re and Pr, both evaluated at the film temperature. No other wall correction is necessary for this formulation. Method is shown without modification in [2] and many other texts.

\[
Nu_D = 0.3 + \frac{0.62 Re^{0.5} Pr^{1/3}}{1 + (0.4/Pr)^{2/3}} \cdot 0.25 \left[1 + \left(\frac{Re_D}{282000}\right)^{5/8}\right]^{0.8}
\]

Parameters

- **Re** [float] Reynolds number with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at film temperature, [-]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

Notes

May underestimate heat transfer in some cases, as it the formula is described in [1] as “appears to provide a lower bound for RePr > 0.4”. An alternate exponent for a smaller range is also presented in [1].

This method applies to both the laminar and turbulent regimes.

References

[1], [2]

Examples

Example 7.3 in [2], matches.

```python
>>> Nu_cylinder_Churchill_Bernstein(6071, 0.7)
40.63708594124974
```

ht.conv_external.Nu_cylinder_Sanitjai_Goldstein (Re, Pr)
Calculates Nusselt number for crossflow across a single tube at a specified Re and Pr, both evaluated at the film temperature. No other wall correction is necessary for this formulation. Method is the most recent implemented here and believed to be more accurate than other formulations available.

\[
Nu = 0.446 Re^{0.5} Pr^{0.35} + 0.528 \left[(6.5 \exp(Re/5000))^{-5} + (0.031 Re^{0.8})^{-5}\right]^{-1/5} Pr^{0.42}
\]

Parameters

- **Re** [float] Reynolds number with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at film temperature, [-]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]
Notes

Developed with test results for water, mixtures of ethylene glycol and water, and air (Pr = 0.7 to 176). Re range from 2E3 to 9E4. Also presents results for local heat transfer coefficients.

This method applies to both the laminar and turbulent regimes.

References

[1]

Examples

```python
>>> Nu_cylinder_Sanitjai_Goldstein(6071, 0.7)
40.38327083519522
```

ht.conv_external.Nu_cylinder_Fand(Re, Pr)
Calculates Nusselt number for crossflow across a single tube at a specified Re and Pr, both evaluated at the film temperature. No other wall correction is necessary for this formulation. Also shown in [2].

\[
Nu = (0.35 + 0.34Re^{0.5} + 0.15Re^{0.58})Pr^{0.3}
\]

Parameters

- **Re** [float] Reynolds number with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at film temperature, [-]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

Notes

Developed with test results for water, and Re from 1E4 to 1E5, but also compared with other data in the literature. Claimed validity of Re from 1E-1 to 1E5.

This method applies to both the laminar and turbulent regimes.

References

[1], [2]

Examples

```python
>>> Nu_cylinder_Fand(6071, 0.7)
45.19984325481126
```

ht.conv_external.Nu_cylinder_Perkins_Leppert_1964(Re, Pr, mu=None, muw=None)
Calculates Nusselt number for crossflow across a single tube as shown in [1] at a specified Re and Pr, both evaluated at the free stream temperature. Recommends a viscosity exponent correction of 0.25, which is applied only if provided. Also shown in [2].

\[
Nu = \left[0.31Re^{0.5} + 0.11Re^{0.67}\right]Pr^{0.4}\left(\frac{\mu}{\mu_w}\right)^{0.25}
\]
Parameters

Re [float] Reynolds number with respect to cylinder diameter, [-]
Pr [float] Prandtl number at free stream temperature, [-]
mu [float, optional] Viscosity of fluid at the free stream temperature [Pa*s]
muw [float, optional] Viscosity of fluid at the wall temperature [Pa*s]

Returns

Nu [float] Nusselt number with respect to cylinder diameter, [-]

Notes

Considers new data since Nu_cylinder_Perkins_Leppert_1962, Re from 2E3 to 1.2E5, Pr from 1 to 7, and surface to bulk temperature differences of 11 to 66.
This method applies to both the laminar and turbulent regimes.

References

[1], [2]

Examples

```python
>>> Nu_cylinder_Perkins_Leppert_1964(6071, 0.7)
53.61767038619986
```

ht.conv_external.Nu_cylinder_Perkins_Leppert_1962(Re, Pr, mu=None, muw=None)
Calculates Nusselt number for crossflow across a single tube as shown in [1] at a specified Re and Pr, both evaluated at the free stream temperature. Recommends a viscosity exponent correction of 0.25, which is applied only if provided. Also shown in [2].

\[
Nu = \left[0.30Re^{0.5} + 0.10Re^{0.67}\right] Pr^{0.4} \left(\frac{\mu}{\mu_w}\right)^{0.25}
\]

Parameters

Re [float] Reynolds number with respect to cylinder diameter, [-]
Pr [float] Prandtl number at free stream temperature, [-]
mu [float, optional] Viscosity of fluid at the free stream temperature [Pa*s]
muw [float, optional] Viscosity of fluid at the wall temperature [Pa*s]

Returns

Nu [float] Nusselt number with respect to cylinder diameter, [-]

Notes

Considered results with Re from 40 to 1E5, Pr from 1 to 300; and viscosity ratios of 0.25 to 4.
This method applies to both the laminar and turbulent regimes.
References

[1], [2]

Examples

```python
>>> Nu_cylinder_Perkins_Leppert_1962(6071, 0.7)
49.97164291175499
```

ht.conv_external.Nu_cylinder_Whitaker(Re, Pr, mu=None, muw=None)
Calculates Nusselt number for crossflow across a single tube as shown in [1] at a specified Re and Pr, both evaluated at the free stream temperature. Recommends a viscosity exponent correction of 0.25, which is applied only if provided. Also shown in [2].

\[ Nu_D = (0.4Re_0^{0.5} + 0.06Re_{0.5}^{2/3})Pr^{0.4} \left( \frac{\mu}{\mu_w} \right)^{0.25} \]

Parameters

- **Re** [float] Reynolds number with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at free stream temperature, [-]
- **mu** [float, optional] Viscosity of fluid at the free stream temperature [Pa*s]
- **muw** [float, optional] Viscosity of fluid at the wall temperature [Pa*s]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

Notes

Developed considering data from 1 to 1E5 Re, 0.67 to 300 Pr, and range of viscosity ratios from 0.25 to 5.2. Found experimental data to generally agree with it within 25%.

This method applies to both the laminar and turbulent regimes.

References

[1], [2]

Examples

```python
>>> Nu_cylinder_Whitaker(6071, 0.7)
45.94527461589126
```

ht.conv_external.Nu_cylinder_McAdams(Re, Pr)
Calculates Nusselt number for crossflow across a single tube at a specified Re and Pr, both evaluated at the film temperature. No other wall correction is necessary for this formulation. Also shown in [2].

\[ Nu = (0.35 + 0.56Re^{0.52})Pr^{0.3} \]

Parameters

- **Re** [float] Reynolds number with respect to cylinder diameter, [-]
Heat Transfer Documentation, Release 0.1.56

**Pr** [float] Prandtl number at film temperature, [-]

**Returns**

**Nu** [float] Nusselt number with respect to cylinder diameter, [-]

**Notes**

Developed with very limited test results for water only.
This method applies to both the laminar and turbulent regimes.

**References**

[1], [2]

**Examples**

```python
>>> Nu_cylinder_McAdams(6071, 0.7)
46.98179235867934
```

ht.conv_external.Nu_external_cylinder(Re, Pr, Prw=None, mu=None, muw=None, Method=None)

Calculates Nusselt number for crossflow across a single tube at a specified Re and Pr according to the specified method. Optional parameters are Prw, mu, and muw. This function has eight methods available. The ‘Sanjitai-Goldstein’ method is the default.

The front of the cylinder is normally always in a laminar regime; whereas the back is turbulent. The proportions change with Re; all correlations take this into account. For this heat transfer case, there is no separation between laminar and turbulent methods.

**Parameters**

- **Re** [float] Reynolds number of fluid with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at either the free stream or wall temperature depending on the method, [-]
- **Prw** [float, optional] Prandtl number at wall temperature, [-]
- **mu** [float, optional] Viscosity of fluid at the free stream temperature [Pa*s]
- **muw** [float, optional] Viscosity of fluid at the wall temperature [Pa*s]

**Returns**

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

**Other Parameters**

- **Method** [string, optional] A string of the function name to use, as in the dictionary conv_external_cylinder_methods.

**Notes**

A comparison of the methods for various Prandtl and Reynolds number ranges is plotted below.
Examples

```python
>>> Nu_external_cylinder(6071, 0.7)
40.38327083519522
```

ht.conv_external.Nu_external_cylinder_methods(Re, Pr, Prw=None, mu=None, 
muw=None, check_ranges=True)

This function returns a list of correlation names for forced convection over an external cylinder.

The preferred method ‘Sanitjai-Goldstein’.

Parameters

- **Re** [float] Reynolds number of fluid with respect to cylinder diameter, [-]
- **Pr** [float] Prandtl number at either the free stream or wall temperature depending on the method, [-]
- **Prw** [float, optional] Prandtl number at wall temperature, [-]
- **mu** [float, optional] Viscosity of fluid at the free stream temperature [Pa*s]
- **muw** [float, optional] Viscosity of fluid at the wall temperature [Pa*s]
- **check_ranges** [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

- **methods** [list[str]] List of methods which can be used to calculate Nu with the given inputs

Examples

```python
>>> Nu_external_cylinder_methods(0.72, 1E7)[0]
'Sanitjai-Goldstein'
```

ht.conv_external.Nu_horizontal_plate_laminar_Baehr(Re, Pr)

Calculates Nusselt number for laminar flow across an isothermal flat plate at a specified Re and Pr, both evaluated at the bulk temperature. No other wall correction is necessary for this formulation. Four different equations are used for different Prandtl number ranges.

The equation for the common Prandtl number range is also recommended in [2] and [3].

if Pr < 0.005:

\[
\text{Nu}_L = 1.128 \text{Re}^{0.5} \text{Pr}^{0.5}
\]

if 0.005 < Pr < 0.05:

\[
\text{Nu}_L = 1.0 \text{Re}^{0.5} \text{Pr}^{0.5}
\]

if 0.6 < Pr < 10:

\[
\text{Nu}_L = 0.664 \text{Re}^{0.5} \text{Pr}^{1/3}
\]

if Pr > 10:

\[
\text{Nu}_L = 0.678 \text{Re}^{0.5} \text{Pr}^{1/3}
\]

Parameters
Re [float] Reynolds number with respect to plate length and bulk fluid properties, [-]

Pr [float] Prandtl number at bulk temperature, [-]

Returns

Nu [float] Nusselt number with respect to plate length and bulk temperature, [-]

Notes

Does not take into account the impact of free convection, which can increase the convection substantially.

References

[1], [2], [3]

Examples

```python
>>> Nu_horizontal_plate_laminar_Baehr(1e5, 0.7)
186.4378528752262
```

ht.conv_external.Nu_horizontal_plate_laminar_Churchill_Ozoe(Re, Pr)

Calculates Nusselt number for laminar flow across an isothermal flat plate at a specified Re and Pr, both evaluated at the bulk temperature. No other wall correction is necessary for this formulation. A single equation covers all Prandtl number ranges.

\[ Nu_L = \frac{0.6774 R_e_L^{1/2} P_r^{1/3}}{[1 + (0.0468/P_r)^{2/3}]^{1/4}} \]

Parameters

Re [float] Reynolds number with respect to plate length and bulk fluid properties, [-]

Pr [float] Prandtl number at bulk temperature, [-]

Returns

Nu [float] Nusselt number with respect to plate length and bulk temperature, [-]

Notes

Does not take into account the impact of free convection, which can increase the convection substantially.

References

[1], [2]

Examples

```python
>>> Nu_horizontal_plate_laminar_Churchill_Ozoe(1e5, 0.7)
183.08600782591418
```
ht.conv_external.Nu_horizontal_plate_turbulent_Schlichting(Re, Pr)
Calculates Nusselt number for turbulent flow across an isothermal flat plate at a specified Re and Pr, both evaluated at the bulk temperature. The formulation of Schlichting is used, which adds a surface friction term to a formulation from Petukhov and Popov.

\[
\text{Nu}_L = \frac{0.037}{1 + 2.443 \text{Re}_L^{-0.1} \text{Pr}^{-2/3}}
\]

**Parameters**
- **Re** [float] Reynolds number with respect to plate length and bulk fluid properties, [-]
- **Pr** [float] Prandtl number at bulk temperature, [-]

**Returns**
- **Nu** [float] Nusselt number with respect to plate length and bulk temperature, [-]

**Notes**
Does not take into account the impact of free convection, which can increase the convection substantially.

**References**
[1], [2]

**Examples**
```python
>>> Nu_horizontal_plate_turbulent_Schlichting(1e5, 0.7)
309.620048541267
```

ht.conv_external.Nu_horizontal_plate_turbulent_Kreith(Re, Pr)
Calculates Nusselt number for turbulent flow across an isothermal flat plate at a specified Re and Pr, both evaluated at the bulk temperature. The formulation of Kreith is used.

\[
\text{Nu}_L = 0.036 \text{Re}_L^{0.8} \text{Pr}^{2/3}
\]

**Parameters**
- **Re** [float] Reynolds number with respect to plate length and bulk fluid properties, [-]
- **Pr** [float] Prandtl number at bulk temperature, [-]

**Returns**
- **Nu** [float] Nusselt number with respect to plate length and bulk temperature, [-]

**Notes**
Does not take into account the impact of free convection, which can increase the convection substantially. Applies for turbulent flow only.

**References**
[1]
Examples

```python
>>> Nu_horizontal_plate_turbulent_Kreith(1.03e6, 0.71)
2074.8740070411122
```

ht.conv_external.Nu_external_horizontal_plate(Re, Pr, L=None, x=None, Method=None, laminar_method='Baehr', turbulent_method='Schlichting', Re_transition=500000.0)

This function calculates the heat transfer coefficient for external forced convection along a horizontal plate. Requires at a minimum a flow’s Reynolds and Prandtl numbers Re and Pr. L and x are not used by any correlations presently, but are included for future support.

If no correlation’s name is provided as Method, the most accurate applicable correlation is selected.

Parameters

- Re [float] Reynolds number with respect to bulk properties and plate length, [-]
- Pr [float] Prandtl number with respect to bulk properties, [-]
- L [float, optional] Length of horizontal plate, [m]
- x [float, optional] Length of horizontal plate for specific calculation distance, [m]

Returns

- Nu [float] Nusselt number with respect to plate length, [-]

Other Parameters

- Method [string, optional] A string of the function name to use, as in the dictionary conv_horizontal_plate_methods
- laminar_method [str, optional] The preferred method for laminar flow, [-]
- turbulent_method [str, optional] The preferred method for turbulent flow, [-]
- Re_transition [float, optional] The transition Reynolds number for laminar changing to turbulent flow, [-]

Examples

Turbulent example

```python
>>> Nu_external_horizontal_plate(Re=1E7, Pr=.7)
11496.95299969829
```

ht.conv_external.Nu_external_horizontal_plate_methods(Re, Pr, L=None, x=None, check_ranges=True)

Returns a list of correlation names for calculating Nusselt number for forced convection across a horizontal plate, supporting both laminar and turbulent regimes.

Parameters

- Re [float] Reynolds number with respect to bulk properties and plate length, [-]
- Pr [float] Prandtl number with respect to bulk properties, [-]
- L [float] Length of horizontal plate, [m]
- x [float] Length of horizontal plate for specific calculation distance, [m]
check_ranges  [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

methods  [list[str]] List of methods which can be used to calculate Nu with the given inputs

Examples

```python
>>> Nu_external_horizontal_plate_methods(Re=1e7, Pr=.7)[0]
'Schlichting'
```

## 2.8 Free convection to immersed bodies (ht.conv_free_immersed)

ht.conv_free_immersed.Nu_vertical_plate_Churchill(Pr, Gr)

Calculates Nusselt number for natural convection around a vertical plate according to the Churchill-Chu [1] correlation, also presented in [2]. Plate must be isothermal; an alternate expression exists for constant heat flux.

\[
N_u L = \left[ 0.825 + \frac{0.387 R_o L^{1/6}}{1 + (0.492/Pr)^9/16} \right]^{2/7}
\]

Parameters

Pr  [float] Prandtl number [-]

Gr  [float] Grashof number [-]

Returns

Nu  [float] Nusselt number with respect to height, [-]

Notes

Although transition from laminar to turbulent is discrete in reality, this equation provides a smooth transition in value from laminar to turbulent. Checked with the original source.

Can be applied to vertical cylinders as well, subject to the criteria below:

\[
\frac{D}{L} \geq \frac{35}{Gr L^{1/4}}
\]

References

[1], [2]

Examples

From [2], Example 9.2, matches:

```python
>>> Nu_vertical_plate_Churchill(0.69, 2.63E9)
147.16185223770603
```
ht.conv_free_immersed.Nu_free_vertical_plate(Pr, Gr, buoyancy=None, H=None, W=None, Method=None)

This function calculates the heat transfer coefficient for external free convection from a vertical plate.

Requires at a minimum a fluid’s Prandtl number $Pr$, and the Grashof number $Gr$ for the system fluid (which require $T$ and $P$ to obtain).

$L$ and $W$ are not used by any correlations presently, but are included for future support.

If no correlation’s name is provided as Method, the ‘Churchill’ correlation is selected.

**Parameters**

- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]
- **H** [float, optional] Height of vertical plate, [m]
- **W** [float, optional] Width of the vertical plate, [m]

**Returns**

- **Nu** [float] Nusselt number with respect to plate height, [-]

**Other Parameters**

- **Method** [string, optional] A string of the function name to use; one of (‘Churchill’, ).

**Examples**

Turbulent example

```python
>>> Nu_free_vertical_plate(0.69, 2.63E9, False)
147.16185223770603
```

ht.conv_free_immersed.Nu_free_vertical_plate_methods(Pr, Gr, H=None, W=None, check_ranges=True)

This function returns a list of methods for calculating heat transfer coefficient for external free convection from a vertical plate.

Requires at a minimum a fluid’s Prandtl number $Pr$, and the Grashof number $Gr$ for the system fluid (which require $T$ and $P$ to obtain).

$L$ and $W$ are not used by any correlations presently, but are included for future support.

**Parameters**

- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]
- **H** [float, optional] Height of vertical plate, [m]
- **W** [float, optional] Width of the vertical plate, [m]
- **check_ranges** [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

**Returns**
methods [list[str]] List of methods which can be used to calculate $Nu$ with the given inputs, [-]

Examples

```python
>>> Nu_free_vertical_plate_methods(0.69, 2.63E9)
['Churchill']
```

$ht.conv_free_immersed.Nu_horizontal_plate_McAdams(Pr, Gr, buoyancy=True)$
Calculates the Nusselt number for natural convection above a horizontal plate according to the McAdams [1] correlations. The plate must be isothermal. Four different equations are used, two each for laminar and turbulent; the two sets of correlations are required because if the plate is hot, buoyancy lifts the fluid off the plate and enhances free convection whereas if the plate is cold, the cold fluid above it settles on it and decreases the free convection.

Parameters

- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]

Returns

- **Nu** [float] Nusselt number with respect to length, [-]

References

[1]

Examples

```python
>>> Nu_horizontal_plate_McAdams(5.54, 3.21e8, buoyancy=True)
181.73121274384457
>>> Nu_horizontal_plate_McAdams(5.54, 3.21e8, buoyancy=False)
55.44564799362829
>>> Nu_horizontal_plate_McAdams(.01, 3.21e8, buoyancy=True)
22.857041558492334
>>> Nu_horizontal_plate_McAdams(.01, 3.21e8, buoyancy=False)
11.428520779246167
```

$ht.conv_free_immersed.Nu_horizontal_plate_VDI(Pr, Gr, buoyancy=True)$
Calculates the Nusselt number for natural convection above a horizontal plate according to the VDI [1] correlations. The plate must be isothermal. Three different equations are used, one each for laminar and turbulent for the heat transfer happening at upper surface case and one for the case of heat transfer happening at the lower surface. The lower surface correlation is recommened for the laminar flow regime. The two different sets of correlations are required because if the plate is hot, buoyancy lifts the fluid off the plate and enhances free convection whereas if the plate is cold, the cold fluid above it settles on it and decreases the free convection.

Parameters

- **Pr** [float] Prandtl number with respect to fluid properties [-]
Gr [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]

buoyancy [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]

Returns

Nu [float] Nusselt number with respect to length, [-]

Notes

The characteristic length suggested for use is as follows, with a and b being the length and width of the plate.

\[ L = \frac{ab}{2(a + b)} \]

The buoyancy enhanced cases are from [2]; the other is said to be from [3], although the equations there are not quite the same and do not include the Prandtl number correction.

References

[1], [2], [3]

Examples

```python
>>> Nu_horizontal_plate_VDI(5.54, 3.21e8, buoyancy=True)
203.89681224927565
>>> Nu_horizontal_plate_VDI(5.54, 3.21e8, buoyancy=False)
39.16864971535617
```

ht.conv_free_immersed.Nu_horizontal_plate_Rohsenow (Pr, Gr, buoyancy=True)

Calculates the Nusselt number for natural convection above a horizontal plate according to the Rohsenow, Hartnett, and Cho (1998) [1] correlations. The plate must be isothermal. Three different equations are used, one each for laminar and turbulent for the heat transfer happening at upper surface case and one for the case of heat transfer happening at the lower surface.

The lower surface correlation is recommended for the laminar flow regime. The two different sets of correlations are required because if the plate is hot, buoyancy lifts the fluid off the plate and enhances free convection whereas if the plate is cold, the cold fluid above it settles on it and decreases the free convection.

Parameters

Pr [float] Prandtl number with respect to fluid properties [-]

Gr [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]

buoyancy [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]

Returns

Nu [float] Nusselt number with respect to length, [-]
Notes

The characteristic length suggested for use is as follows, with $a$ and $b$ being the length and width of the plate.

$$L = \frac{ab}{2(a+b)}$$

References

[1]

Examples

```python
>>> Nu_horizontal_plate_Rohsenow(5.54, 3.21e8, buoyancy=True)
175.91054716322836
>>> Nu_horizontal_plate_Rohsenow(5.54, 3.21e8, buoyancy=False)
35.95799244863986
```

This function calculates the heat transfer coefficient for external free convection from a horizontal plate.

Requires at a minimum a fluid’s Prandtl number $Pr$, and the Grashof number $Gr$ for the system fluid, temperatures, and geometry.

$L$ and $W$ are not used by any correlations presently, but are included for future support.

If no correlation’s name is provided as Method, the ‘VDI’ correlation is selected.

Parameters

- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]
- **L** [float, optional] Length of horizontal plate, [m]
- **W** [float, optional] Width of the horizontal plate, [m]

Returns

- **Nu** [float] Nusselt number with respect to plate length, [-]

Other Parameters

- **Method** [string, optional] A string of the function name to use, as in the dictionary `conv_free_horizontal_plate_methods`

Examples

Turbulent example

```python
>>> Nu_free_horizontal_plate(5.54, 3.21e8, buoyancy=True)
203.89681224927565
```
Nu_free_horizontal_plate(5.54, 3.21e8, buoyancy=True, Method='McAdams')

181.73121274384457

ht.conv_free_immersed.Nu_free_horizontal_plate_methods(Pr, Gr, buoyancy, L=None, W=None, check_ranges=True)

This function returns a list of methods for calculating heat transfer coefficient for external free convection from a vertical plate.

Requires at a minimum a fluid’s Prandtl number Pr, and the Grashof number Gr for the system fluid, temperatures, and geometry.

$L$ and $W$ are not used by any correlations presently, but are included for future support.

 Parameters
- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - fluid temperature difference [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]
- **L** [float, optional] Length of horizontal plate, [m]
- **W** [float, optional] Width of the horizontal plate, [m]
- **check_ranges** [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

 Returns
- **methods** [list[str]] List of methods which can be used to calculate $Nu$ with the given inputs, [-]

Examples

>>> Nu_free_horizontal_plate_methods(0.69, 2.63E9, True)

['VDI', 'McAdams', 'Rohsenow']

ht.conv_free_immersed.Nu_sphere_Churchill(Pr, Gr)

Calculates Nusselt number for natural convection around a sphere according to the Churchill [1] correlation. Sphere must be isothermal.

$$Nu_D = 2 + \frac{0.589 Ra_D^{1/4}}{[1 + (0.469/Pr)^{9/16}]^{1/9}} \cdot \left(1 + \frac{7.44 \times 10^{-8} Ra}{[1 + (0.469/Pr)^{9/16}]^{16/9}}\right)^{1/12}$$

 Parameters
- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number [-]

 Returns
- **Nu** [float] Nusselt number, [-]
Notes

Although transition from laminar to turbulent is discrete in reality, this equation provides a smooth transition in value from laminar to turbulent. Checked with the original source.

Good for Ra < 1E13. Limit of Nu is 2 at low Grashof numbers.

References

[1]

Examples

```python
>>> Nu_sphere_Churchill(.7, 1E7)
25.670869440317578
```

ht.conv_free_immersed.Nu_vertical_cylinder_Griffiths_Davis_Morgan(Pr, Gr, turbulent=None)

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] correlated by [2], as presented in [3] and [4].

\[
Nu_H = 0.67Ra_H^{0.25}, \quad 10^7 < Ra < 10^9
\]

\[
Nu_H = 0.0782Ra_H^{0.357}, \quad 10^9 < Ra < 10^{11}
\]

Parameters

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number [-]
- **turbulent** [bool or None, optional]
  
  Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection

Returns

- **Nu** [float] Nusselt number, [-]

Notes

Cylinder of diameter 17.43 cm, length from 4.65 to 263.5 cm. Air as fluid. Transition between ranges is not smooth. If outside of range, no warning is given.

References

[1], [2], [3], [4]

Examples
Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] correlated by [2], as presented in [3] and [4].

\[ Nu_H = 0.555Ra_H^{0.25}, \quad 10^4 < Ra < 10^8 \]

\[ Nu_H = 0.129Ra_H^{1/3}, \quad 10^8 < Ra < 10^{12} \]

**Parameters**

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number [-]
- **turbulent** [bool or None, optional]

*Whether or not to force the correlation to return the turbulent* result; will return the laminar regime if False; leave as None for automatic selection

**Returns**

- **Nu** [float] Nusselt number, [-]

**Notes**

Cylinder of diameter 3.5 cm, length from L/D = 4.3. Air as fluid. Transition between ranges is not smooth. If outside of range, no warning is given. Results are presented rounded in [4], and the second range is not shown in [3].

**References**

[1], [2], [3], [4]

**Examples**

```python
>>> Nu_vertical_cylinder_Jakob_Linke_Morgan(.7, 2E10)
310.90835207860454
```

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] correlated by [2], as presented in [3] and [4].

\[ Nu_H = 1.07Ra_H^{0.28}, \quad 2 \times 10^6 < Ra < 2 \times 10^8 \]

\[ Nu_H = 0.152Ra_H^{0.38}, \quad 2 \times 10^8 < Ra < 2 \times 10^{11} \]

**Parameters**

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number [-]
turbulent [bool or None, optional]

Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection

Returns

Nu [float] Nusselt number, [-]

Notes

Cylinder of diameters 0.475 cm to 7.62 cm, L/D from 8 to 127. Isothermal boundary condition was assumed, but not verified. Transition between ranges is not smooth. If outside of range, no warning is given. The higher range of [1] is not shown in [3], and the formula for the first is actually for the second in [3].

References

[1], [2], [3], [4]

Examples

```python
>>> Nu_vertical_cylinder_Carne_Morgan(.7, 2E8)
204.31470629065677
```

ht.conv_free_immersed.Nu_vertical_cylinder_Eigenson_Morgan(Pr, Gr, turbulent=None)

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] correlated by [2], presented in [3] and in more detail in [4].

\[ Nu_H = 0.48R_{a_H}^{0.25}, \quad 10^9 < Ra \]

\[ Nu_H = 51.5 + 0.0000726R_{a_H}^{0.63}, \quad 10^9 < Ra < 1.69 \times 10^{10} \]

\[ Nu_H = 0.148R_{a_H}^{1/3} - 127.6, \quad 1.69 \times 10^{10} < Ra \]

Parameters

Pr [float] Prandtl number [-]

Gr [float] Grashof number [-]

turbulent [bool or None, optional]

Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection

Returns

Nu [float] Nusselt number, [-]
Notes

Author presents results as appropriate for both flat plates and cylinders. Height of 2.5 m with diameters of 2.4, 7.55, 15, 35, and 50 mm. Another experiment of diameter 58 mm and length of 6.5 m was considered. Cylinder of diameters 0.475 cm to 7.62 cm, L/D from 8 to 127. Transition between ranges is not smooth. If outside of range, no warning is given. Formulas are presented similarly in [3] and [4], but only [4] shows the transition formula.

References

[1], [2], [3], [4]

Examples

```python
>>> Nu_vertical_cylinder_Eigenson_Morgan(0.7, 2E10)
230.55946525499715
```

ht.conv_free_immersed.Nu_vertical_cylinder_Touloukian_Morgan(Pr, Gr, turbulent=None)

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] correlated by [2], as presented in [3] and [4].

\[
\begin{align*}
Nu_H &= 0.726 Ra_H^{0.25}, \quad 2 \times 10^8 < Ra < 4 \times 10^{10} \\
Nu_H &= 0.0674 (Gr_H Pr^{1.29})^{1/3}, \quad 4 \times 10^{10} < Ra < 9 \times 10^{11}
\end{align*}
\]

Parameters

- `Pr` [float] Prandtl number [-]
- `Gr` [float] Grashof number [-]
- `turbulent` [bool or None, optional]

Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection

Returns

- `Nu` [float] Nusselt number, [-]

Notes

Cylinder of diameters 2.75 inch, with heights of 6, 18, and 36.25 inch. Temperature was controlled via multiple separately controlled heating sections. Fluids were water and ethylene-glycol. Transition between ranges is not smooth. If outside of range, no warning is given. [2], [3], and [4] are in complete agreement about this formulation.

References

[1], [2], [3], [4]
Examples

```python
>>> Nu_vertical_cylinder_Touloukian_Morgan(.7, 2E10)
249.72879961097854
```

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] and [2] correlated by [3], as presented in [4], [5], and [6].

\[
Nu_H = 0.59Ra_H^{0.25}, \quad 10^4 < Ra < 10^9
\]

\[
Nu_H = 0.13Ra_H^{1/3}, \quad 10^9 < Ra < 10^{12}
\]

Parameters

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number [-]

  turbulent [bool or None, optional]

  Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection.

Returns

- **Nu** [float] Nusselt number, [-]

Notes

Transition between ranges is not smooth. If outside of range, no warning is given. For ranges under 10^4, a graph is provided, not included here.

References

[1], [2], [3], [4], [5], [6]

Examples

```python
>>> Nu_vertical_cylinder_McAdams_Weiss_Saunders(.7, 2E10)
313.31849434277973
```

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] and [2] correlated by [3], as presented in [4], [5], and [6].

\[
Nu_H = 0.59Ra_H^{0.25}, \quad 10^4 < Ra < 10^9
\]

\[
Nu_H = 0.13Ra_H^{1/3}, \quad 10^9 < Ra < 10^{12}
\]

Parameters

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number [-]

  turbulent [bool or None, optional]

  Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection.

Returns

- **Nu** [float] Nusselt number, [-]
Prandtl number [-]

Grashof number [-]

Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection

Returns

Nusselt number, [-]

Notes

Transition between ranges is not smooth. If outside of range, no warning is given.

References

[1], [2], [3], [4], [5]

Examples

```python
>>> Nu_vertical_cylinder_Kreith_Eckert(.7, 2E10)
240.2593473033196
```

ht.conv_free_immersed.Nu_vertical_cylinder_Hanesian_Kalish_Morgan(Pr, Gr)

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to the results of [1] correlated by [2], also as presented in [3] and [4].

\[ Nu_H = 0.48Ra_H^{0.23}, \quad 10^6 < Ra < 10^8 \]

Parameters

Pr [float] Prandtl number [-]

Gr [float] Grashof number [-]

Returns

Nu [float] Nusselt number, [-]

Notes

For air and fluoro-carbons. If outside of range, no warning is given. Laminar range only!

References

[1], [2], [3], [4]
Examples

```python
>>> Nu_vertical_cylinder_Hanesian_Kalish_Morgan(.7, 1E7)
18.014150492696604
```

ht.conv_free_immersed.Nu_vertical_cylinder_Al_Arabi_Khamis(Pr, Gr, L, D, turbulent=None)

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to [1], also as presented in [2] and [3].

\[
Nu_H = 2.9 Ra_H^{0.25} / Gr_D^{1/12}, \quad 9.88 \times 10^7 \leq Ra_H \leq 2.7 \times 10^9 \\
Nu_H = 0.47 Ra_H^{0.333} / Gr_D^{1/12}, \quad 2.7 \times 10^9 \leq Ra_H \leq 2.95 \times 10^{10}
\]

Parameters

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number with respect to cylinder height [-]
- **L** [float] Length of vertical cylinder, [m]
- **D** [float] Diameter of cylinder, [m]
- **turbulent** [bool or None, optional]
  
  Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False; leave as None for automatic selection

Returns

- **Nu** [float] Nusselt number, [-]

Notes

For air. Local Nusselt number results also given in [1]. D from 12.75 to 51 mm; H from 300 to 2000 mm. Temperature kept constant by steam condensing.

If outside of range, no warning is given. Applies for range of:

\[
1.08 \times 10^4 \leq Gr_D \leq 6.9 \times 10^5
\]

References

[1], [2], [3]

Examples

```python
>>> Nu_vertical_cylinder_Al_Arabi_Khamis(.71, 2E10, 10, 1)
280.39793209114765
```

ht.conv_free_immersed.Nu_vertical_cylinder_Popiel_Churchill(Pr, Gr, L, D)

Calculates Nusselt number for natural convection around a vertical isothermal cylinder according to [1], also presented in [2].

\[
\frac{Nu}{Nu_{L,fp}} = 1 + B \left[ 32^{0.5} Gr \frac{0.25 L}{D} \right]^C
\]
\[ B = 0.0571322 + 0.20305Pr^{-0.43} \]
\[ C = 0.9165 - 0.0043Pr^{0.5} + 0.01333\ln Pr + 0.0004809/Pr \]

Parameters

- \( Pr \) [float] Prandtl number [-]
- \( Gr \) [float] Grashof number with respect to cylinder height [-]
- \( L \) [float] Length of vertical cylinder, [m]
- \( D \) [float] Diameter of cylinder, [m]

Returns

- \( Nu \) [float] Nusselt number, [-]

Notes

For \( 0.01 < Pr < 100 \). Requires a vertical flat plate correlation. Both [2], [3] present a power of 2 instead of 0.5 on the 32 in the equation, but the original has the correct form.

References

[1], [2], [3]

Examples

```>>> Nu_vertical_cylinder_Popiel_Churchill(0.7, 1E10, 2.5, 1) 228.89790055149896```

ht.conv_free_immersed.Nu_vertical_cylinder(\( Pr, Gr, L=None, D=None, Method=None \))

This function handles choosing which vertical cylinder free convection correlation is used. Generally this is used by a helper class, but can be used directly. Will automatically select the correlation to use if none is provided; returns None if insufficient information is provided.

Preferred functions are ‘Popiel & Churchill’ for fully defined geometries, and ‘McAdams, Weiss & Saunders’ otherwise.

Parameters

- \( Pr \) [float] Prandtl number [-]
- \( Gr \) [float] Grashof number with respect to cylinder height [-]
- \( L \) [float, optional] Length of vertical cylinder, [m]
- \( D \) [float, optional] Diameter of cylinder, [m]

Returns

- \( Nu \) [float] Nusselt number, [-]

Other Parameters

- \( Method \) [string, optional] A string of the function name to use, as in the dictionary vertical_cylinder_correlations
Examples

```python
>>> Nu_vertical_cylinder(0.72, 1E7)
30.562236756513943
```

`ht.conv_free_immersed.Nu_vertical_cylinder_methods(Pr, Gr, L=None, D=None, check_ranges=True)`

This function returns a list of correlation names for free convection to a vertical cylinder.

The functions returned are ‘Popiel & Churchill’ for fully defined geometries, and ‘McAdams, Weiss & Saunders’ otherwise.

Parameters

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number with respect to cylinder height [-]
- **L** [float, optional] Length of vertical cylinder, [m]
- **D** [float, optional] Diameter of cylinder, [m]
- **check_ranges** [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

- **methods** [list[str]] List of methods which can be used to calculate Nu with the given inputs

Examples

```python
>>> Nu_vertical_cylinder_methods(0.72, 1E7)[0]
'McAdams, Weiss & Saunders'
```

`ht.conv_free_immersed.Nu_horizontal_cylinder_Churchill_Chu(Pr, Gr)`

Calculates Nusselt number for natural convection around a horizontal cylinder according to the Churchill-Chu [1] correlation, also presented in [2]. Cylinder must be isothermal; an alternate expression exists for constant heat flux.

\[
Nu_D = \left[ 0.60 + \frac{0.387Ra^{1/6}}{1 + (0.559/Pr)^{3/16}18/27} \right]^2
\]

Parameters

- **Pr** [float] Prandtl number [-]
- **Gr** [float] Grashof number with respect to cylinder diameter, [-]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

Notes

Although transition from laminar to turbulent is discrete in reality, this equation provides a smooth transition in value from laminar to turbulent. Checked with the original source, which has its powers unsimplified but is equivalent.


88 Chapter 2. API Reference
References

[1], [2]

Examples

From [2], Example 9.2, matches:

```python
>>> Nu_horizontal_cylinder_Churchill_Chu(0.69, 2.63E9)
139.13493970073597
```

ht.conv_free_immersed.Nu_horizontal_cylinder_Kuehn_Goldstein(Pr, Gr)
Calculates Nusselt number for natural convection around a horizontal cylinder according to the Kuehn-Goldstein [1] correlation, also shown in [2]. Cylinder must be isothermal.

\[
\frac{2}{Nu_D} = \ln \left[ 1 + \frac{2}{\left\{ 0.518Ra_D^{0.25} \left[ 1 + \left( \frac{0.559}{Pr} \right)^{3/5} \right]^{5/12} \right\}^{15} + \left( 0.1Ra_D^{1/3} \right)^{15}} \right]^{1/15}
\]

Parameters

- **Pr** [float] Prandtl number with respect to film temperature [-]
- **Gr** [float] Grashof number with respect to cylinder diameter, [-]

Returns

- **Nu** [float] Nusselt number with respect to cylinder diameter, [-]

Notes

[1] suggests this expression is valid for all cases except low-Pr fluids. [2] suggests no restrictions.

References

[1], [2]

Examples

```python
>>> Nu_horizontal_cylinder_Kuehn_Goldstein(0.69, 2.63E9)
122.99323525628186
```

ht.conv_free_immersed.Nu_horizontal_cylinder_Morgan(Pr, Gr)
Calculates Nusselt number for natural convection around a horizontal cylinder according to the Morgan [1] correlations, a product of a very large review of the literature. Sufficiently common as to be shown in [2]. Cylinder must be isothermal.

\[
Nu_D = CRa_D^n
\]
<table>
<thead>
<tr>
<th>Gr min</th>
<th>Gr max</th>
<th>C</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E-10</td>
<td>1E-2</td>
<td>0.675</td>
<td>0.058</td>
</tr>
<tr>
<td>1E-2</td>
<td>1E2</td>
<td>1.02</td>
<td>0.148</td>
</tr>
<tr>
<td>1E2</td>
<td>1E4</td>
<td>0.850</td>
<td>0.188</td>
</tr>
<tr>
<td>1E4</td>
<td>1E7</td>
<td>0.480</td>
<td>0.250</td>
</tr>
<tr>
<td>1E7</td>
<td>1E12</td>
<td>0.125</td>
<td>0.333</td>
</tr>
</tbody>
</table>

Parameters

- Pr [float] Prandtl number with respect to film temperature [-]
- Gr [float] Grashof number with respect to cylinder diameter, [-]

Returns

- Nu [float] Nusselt number with respect to cylinder diameter, [-]

Notes

Most comprehensive review with a new proposed equation to date. Discontinuous among the jumps in range. Blindly runs outside if upper and lower limits without warning.

References

[1], [2]

Examples

```python
>>> Nu_horizontal_cylinder_Morgan(0.69, 2.63E9)
151.3881997228419
```

`ht.conv_free_immersed.Nu_horizontal_cylinder(Pr, Gr, Method=None)`

This function handles choosing which horizontal cylinder free convection correlation is used. Generally this is used by a helper class, but can be used directly. Will automatically select the correlation to use if none is provided; returns None if insufficient information is provided.

Preferred functions are ‘Morgan’ when discontinuous results are acceptable and ‘Churchill-Chu’ otherwise.

Parameters

- Pr [float] Prandtl number with respect to film temperature [-]
- Gr [float] Grashof number with respect to cylinder diameter, [-]

Returns

- Nu [float] Nusselt number with respect to cylinder diameter, [-]

Other Parameters

- Method [string, optional] A string of the function name to use, as in the dictionary horizontal_cylinder_correlations
Notes

All fluid properties should be evaluated at the film temperature, the average between the outer surface temperature of the solid, and the fluid temperature far away from the heat transfer interface - normally the same as the temperature before any cooling or heating occurs.

\[ T_f = \frac{(T_{\text{surface}} + T_{\infty})}{2} \]

Examples

```python
>>> Nu_horizontal_cylinder(0.72, 1E7)
24.864192615468973
```

ht.conv_free_immersed.Nu_horizontal_cylinder_methods (Pr, Gr, check_ranges=True)

This function returns a list of correlation names for free convection to a horizontal cylinder.

Preferred functions are ‘Morgan’ when discontinuous results are acceptable and ‘Churchill-Chu’ otherwise.

Parameters

**Pr** [float] Prandtl number with respect to film temperature [-]

**Gr** [float] Grashof number with respect to cylinder diameter, [-]

**check_ranges** [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

**methods** [list[str]] List of methods which can be used to calculate \( Nu \) with the given inputs

Examples

```python
>>> Nu_horizontal_cylinder_methods(0.72, 1E7)[0]
'Morgan'
```

ht.conv_free_immersed.Nu_coil_Xin_Ebadian (Pr, Gr, horizontal=False)

Calculates Nusselt number for natural convection around a vertical or horizontal helical coil suspended in a fluid without forced convection.

For horizontal cases:

\[ N u_D = 0.318 R a_D^{0.293}, \ 5 \times 10^3 < R a < 1 \times 10^5 \]

For vertical cases:

\[ N u_D = 0.290 R a_D^{0.293}, \ 5 \times 10^3 < R a < 1 \times 10^5 \]

Parameters

**Pr** [float] Prandtl number calculated with the film temperature - wall and temperature very far from the coil average, [-]

**Gr** [float] Grashof number calculated with the film temperature - wall and temperature very far from the coil average, and using the outer diameter of the coil [-]

**horizontal** [bool, optional] Whether the coil is horizontal or vertical, [-]

Returns

**Nu** [float] Nusselt number using the outer diameter of the coil and the film temperature, [-]
Notes

This correlation is also reviewed in [2].

References

[1], [2]

Examples

```python
>>> Nu_coil_Xin_Ebadian(0.7, 2E4, horizontal=False)
4.755689726250451
>>> Nu_coil_Xin_Ebadian(0.7, 2E4, horizontal=True)
5.2148597687849785
```

2.9 Free convection to enclosed bodies (ht.conv_free_enclosed)

ht.conv_free_enclosed.Nu_Nusselt_Rayleigh_Holling_Herwig (Pr, Gr, buoyancy=True)  
Calculates the Nusselt number for natural convection between two theoretical flat horizontal plates. The height  
between the plates is infinite, and one of the other dimensions of the plates is much larger than the other.  
This correlation is for the horizontal plate Rayleigh-Benard classic heat transfer problem, not for real finite  
gometry plates.

This model is a non-linear equation which is solved numerically. The model can calculate Nu for Ra ranges  
between 350 and larger numbers; [1] recommends $10^5 < Ra < 10^{15}$.

$$
Nu = \frac{Ra^{1/3}}{0.05 \log\left(\frac{0.008}{16} Ra^{1.323} + 2D\right)^{4/3}}
$$

$$
D = -\frac{14.94}{Ra^{0.25}} + 3.43
$$

Parameters

- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - plate temperature difference [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]

Returns

- **Nu** [float] Nusselt number with respect to height between the two plates, [-]

Notes

A range of calculated values are provided in [1]; they all match the results of this function. This model is  
recommended in [2].

For $Ra < 1708$, $Nu = 1$; for cases not assisted by buoyancy, $Nu$ is also 1.

No success has been found finding an analytical solution in the major CAS packages, but the nonlinear function  
is in fact a function of one variable; this means a pade or chebyshev expansion could be performed.
References

[1], [2]

Examples

```python
>>> Nu_Nusselt_Rayleigh_Holling_Herwig(5.54, 3.21e8, buoyancy=True)
77.54656801896913
```

ht.conv_free_enclosed.Nu_Nusselt_Rayleigh_Probert (Pr, Gr, buoyancy=True)

Calculates the Nusselt number for natural convection between two theoretical flat plates. The height between
the plates is infinite, and one of the other dimensions of the plates is much larger than the other.

This correlation is for the horizontal plate Rayleigh-Benard classic heat transfer problem, not for real finite
geometry plates.

Two sets of equations are used.

For the laminar regime $1708 < Ra \leq 2.2 \times 10^4$:

$$Nu = 0.208(Ra)^{0.25}$$

For the turbulent regime $2.2 \times 10^4 < Ra$:

$$Nu = 0.092(Ra)^{1/3}$$

Parameters

- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - plate temperature differ-
  ence [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot
  plate) or not, [-]

Returns

- **Nu** [float] Nusselt number with respect to height between the two plates, [-]

Notes

This model is recommended in [2] as a rough model.

For $Ra < 1708$, $Nu = 1$; for cases not assisted by **buoyancy**, $Nu$ is also 1.

References

[1], [2]

Examples

```python
>>> Nu_Nusselt_Rayleigh_Probert(5.54, 3.21e8, buoyancy=True)
111.46181048289132
```
ht.conv_free_enclosed.Nu_Nusselt_Rayleigh_Hollands(Pr, Gr, buoyancy=True, Rac=1708)

Calculates the Nusselt number for natural convection between two theoretical flat horizontal plates using the Hollands [1] correlation recommended in [2]. This correlation supports different aspect ratios, so the plates can be real, finite objects and have their heat transfer accurately modeled. The influence comes from the Rac term, which should be calculated separately, using Rac_Nusselt_Rayleigh or Rac_Nusselt_Rayleigh_disk.

\[
Nu = 1 + \left[ 1 - \frac{1708}{Ra} \right]^* \left[ k_1 + 2 \left( \frac{Ra^{1/3}}{k_2} \right)^{1-\log(Ra^{1/3}/k_2)} \right]^* + \left[ \left( \frac{Ra}{5803} \right)^{1/3} - 1 \right]^* \\
\]

\[
k_1 = \frac{1.44}{1 + 0.018/Pr + 0.00136/Pr^2} \\
k_2 = 75 \exp(1.5Pr^{-0.5})
\]

**Parameters**
- **Pr** [float] Prandtl number with respect to fluid properties [-]
- **Gr** [float] Grashof number with respect to fluid properties and plate - plate temperature difference [-]
- **buoyancy** [bool, optional] Whether or not the plate’s free convection is buoyancy assisted (hot plate) or not, [-]
- **Rac** [float, optional] Critical Rayleigh number, [-]

**Returns**
- **Nu** [float] Nusselt number with respect to height between the two plates, [-]

**Notes**
For \( Ra < Ra_c \), \( Nu = 1 \); for cases not assisted by buoyancy, \( Nu \) is also 1.

**References**
[1], [2]

**Examples**

```python
>>> Nu_Nusselt_Rayleigh_Hollands(5.54, 3.21e8, buoyancy=True)
69.02668649510164
```

Plates - 1 m height, 2 m long, 0.2 m long vs a 1 m^3 cube

```python
>>> Nu_Nusselt_Rayleigh_Hollands(0.7, 3.21e6, buoyancy=True, Rac=Rac_Nusselt_Rayleigh(H=1, L=2, W=.2, insulated=False))
4.666249131876477
```

```python
>>> Nu_Nusselt_Rayleigh_Hollaleighs(0.7, 3.21e6, buoyancy=True, Rac=Rac_Nusselt_Rayleigh(H=1, L=1, W=1, insulated=False))
8.786362614129537
```
ht.conv_free_enclosed.Rac_Nusselt_Rayleigh \((H, L, W, insulated=\text{True})\)
Calculates the critical Rayleigh number for free convection to begin in the Nusselt-Rayleigh parallel horizontal plate scenario. There are actually two cases - one for the top plate to be insulated (adiabatic) and the other where it has infinite thermal conductivity/is infinitely thin or not present (perfectly conducting). All real cases will lie between the two.

Parameters

\(H\) [float] Distance between the two plates, [m]
\(L\) [float] Length of the plates, [m]
\(W\) [float] Width of the plates, [m]
\(\text{insulated}\) [bool, optional] Whether the top plate is insulated or uninsulated, [-]

Returns

\(Rac\) [float] Critical Rayleigh number, [-]

Notes

Splines have been fit to data in [1] for the uninsulated case and [2] for the insulated case. The data is presented in the original papers and in [3].

References

[1], [2], [3]

Examples

```python
>>> Rac_Nusselt_Rayleigh(1, .5, 2, False)
2530.500000000005
>>> Rac_Nusselt_Rayleigh(1, .5, 2, True)
2071.0089443385655
```

ht.conv_free_enclosed.Rac_Nusselt_Rayleigh_disk \((H, D, insulated=\text{True})\)
Calculates the critical Rayleigh number for free convection to begin in the parallel horizontal disk scenario. There are actually two cases - one for the top plate to be insulated (adiabatic) and the other where it has infinite thermal conductivity/is infinitely thin or not present (perfectly conducting). All real cases will lie between the two.

Parameters

\(H\) [float] Distance between the two disks, [m]
\(D\) [float] Diameter of the two disks, [m]
\(\text{insulated}\) [bool, optional] Whether the top plate is insulated or uninsulated, [-]

Returns

\(Rac\) [float] Critical Rayleigh number, [-]
Notes

The range of data covered by this function is \( DIH \) from 0.4 to infinity. As infinity is not well suited to polynomial form, the upper limit is 6 in actuality. Values outside that range are rounded to the limits.

This function provides 17-coefficient polynomial fits to interpolate in the table of values in [1]. The source of the coefficients is cited as being from [2].

References

[1], [2]

Examples

```python
>>> Rac_Nusselt_Rayleigh_disk(H=1, D=.4, insulated=False)
151199.9999999945
```

```python
>>> Rac_Nusselt_Rayleigh_disk(H=1, D=4, insulated=False)
1891.520931853363
```

```python
>>> Rac_Nusselt_Rayleigh_disk(2, 1, True)
24347.31479211917
```

```
ht.conv_free_enclosed.Nu_Nusselt_vertical_Thess (Pr, Gr, H=None, L=None)
Calculates the Nusselt number for natural convection between two theoretical vertical flat plates using the correlation by Thess [1] in [1]. This is a variant on the horizontal Rayleigh-Benard classic heat transfer problem. This correlation supports different aspect ratios, so the plates can be real, finite objects and have their heat transfer accurately modeled. The recommended range of the correlation is \( H/L < 80 \).

For \( 1e4 < Ra < 1e7 \):

\[
Nu = 0.42Pr^{0.012}Ra^{0.25} \left( \frac{H}{L} \right)^{-0.25}
\]

For \( 1e7 < Ra > 1e9 \) (or when geometry is unknown):

\[
Nu = 0.049Ra^{0.33}
\]

Parameters

- \( Pr \) [float] Prandtl number with respect to fluid properties [-]
- \( Gr \) [float] Grashof number with respect to fluid properties and plate - plate temperature difference [-]
- \( H \) [float, optional] Height of vertical plate, [m]
- \( L \) [float, optional] Length of vertical plate, [m]

Returns

- \( Nu \) [float] Nusselt number with respect to distance between the two plates, [-]

References

[1]
Examples

```python
>>> Nu_Nusselt_vertical_Thess(.7, 3.21e6)
6.112587569602785
```

```python
>>> Nu_Nusselt_vertical_Thess(.7, 3.21e6, L=10, H=1)
28.79328626041646
```

**ht.conv_free_enclosed.Nu_vertical_helical_coil_Ali**(Pr, Gr)
Calculates Nusselt number for natural convection around a vertical helical coil inside a tank or other vessel according to the Ali [1] correlation.

\[
Nu_L = 0.555Gr^{0.301}Pr^{0.314}
\]

**Parameters**

- **Pr** [float] Prandtl number of the fluid surrounding the coil with properties evaluated at bulk conditions or as described in the notes [-]
- **Gr** [float] Prandtl number of the fluid surrounding the coil with properties evaluated at bulk conditions or as described in the notes (for the two temperatures, use the average coil fluid temperature and the temperature of the fluid outside the coil) [-]

**Returns**

- **Nu** [float] Nusselt number with respect to the total length of the helical coil (and bulk thermal conductivity), [-]

**Notes**

In [1], the temperature at which the fluid surrounding the coil’s properties were evaluated at was calculated in an unusual fashion. The average temperature of the fluid inside the coil \((T_{in} + T_{out})/2\) is averaged with the fluid outside the coil’s temperature.

The correlation is valid for Prandtl numbers between 4.4 and 345, and tank diameter/coil outer diameter ratios between 10 and 30.

**References**

[1]

**Examples**

```python
>>> Nu_vertical_helical_coil_Ali(4.4, 1E11)
1808.5774997297106
```

**ht.conv_free_enclosed.Nu_vertical_helical_coil_Prabhanjan_Rennie_Raghavan**(Pr, Gr)
Calculates Nusselt number for natural convection around a vertical helical coil inside a tank or other vessel according to the Prabhanjan, Rennie, and Raghavan [1] correlation.

\[
Nu_H = 0.0749Ra_H^{0.3421}
\]

The range of Rayleigh numbers is as follows:

\[
9 \times 10^9 < Ra < 4 \times 10^{11}
\]

2.9. Free convection to enclosed bodies (ht.conv_free_enclosed)
Parameters

- **Pr** [float] Prandtl number calculated with the film temperature - wall and temperature very far from the coil average, [-]

- **Gr** [float] Grashof number calculated with the film temperature - wall and temperature very far from the coil average, and using the total height of the coil [-]

Returns

- **Nu** [float] Nusselt number using the total height of the coil and the film temperature, [-]

Notes

[1] also has several other equations using different characteristic lengths.

References

[1]

Examples

```python
>>> nu_vertical_helical_coil_prabhanjan_rennie_raghavan(4.4, 1E11)
720.6211067718227
```

2.10 Internal convection (ht.conv_internal)

ht.conv_internal.laminar_T_const()

Returns internal convection Nusselt number for laminar flows in pipe according to [1], [2] and [3]. Wall temperature is assumed constant. This is entirely theoretically derived and reproduced experimentally.

\[ Nu = 3.66 \]

Returns

- **Nu** [float] Nusselt number, [-]

Notes

This applies only for fully thermally and hydraulically developed and flows.

References

[1], [2], [3]

ht.conv_internal.laminar_Q_const()

Returns internal convection Nusselt number for laminar flows in pipe according to [1], [2], and [3]. Heat flux is assumed constant. This is entirely theoretically derived and reproduced experimentally.

\[ Nu = 4.354 \]

Returns

- **Nu** [float] Nusselt number, [-]
Notes

This applies only for fully thermally and hydraulically developed and flows. Many sources round to 4.36, but [3] does not.

References

[1], [2], [3]

ht.conv_internal.laminar_entry_thermal_Hausen (Re, Pr, L, Di)
Calculates average internal convection Nusselt number for laminar flows in pipe during the thermal entry region according to [1] as shown in [2] and cited by [3].

\[
Nu_D = 3.66 + \frac{0.0668 \frac{D}{L} Re_D Pr}{1 + 0.04 \left( \frac{D}{L} Re_D Pr \right)^{2/3}}
\]

Parameters

Re [float] Reynolds number, [-]
Pr [float] Prandtl number, [-]
L [float] Length of pipe [m]
Di [float] Diameter of pipe [m]

Returns

Nu [float] Nusselt number, [-]

Notes

If Pr >> 1, (5 is a common requirement) this equation also applies to flows with developing velocity profile. As L gets larger, this equation becomes the constant-temperature Nusselt number.

References

[1], [2], [3]

Examples

```python
>>> laminar_entry_thermal_Hausen(Re=100000, Pr=1.1, L=5, Di=0.5)
39.01352358988535
```

ht.conv_internal.laminar_entry_Seider_Tate (Re, Pr, L, Di, mu=None, mu_w=None)
Calculates average internal convection Nusselt number for laminar flows in pipe during the thermal entry region as developed in [1], also shown in [2].

\[
Nu_D = 1.86 \left( \frac{D}{L} Re_D Pr \right)^{1/3} \left( \frac{\mu_b}{\mu_s} \right)^{0.14}
\]

Parameters

Re [float] Reynolds number, [-]
Pr [float] Prandtl number, [-]
L [float] Length of pipe [m]
Di [float] Diameter of pipe [m]
mu [float, optional] Viscosity of fluid, [Pa*s]
mu_w [float, optional] Viscosity of fluid at wall temperature, [Pa*s]

Returns
Nu [float] Nusselt number, [-]

Notes
Reynolds number should be less than 10000. This should be calculated using pipe diameter. Prandtl number should be no less than air and no more than liquid metals; 0.7 < Pr < 16700 Viscosities should be the bulk and surface properties; they are optional. Outside the boundaries, this equation is provides very false results.

References
[1], [2]

Examples

```python
>>> laminar_entry_Seider_Tate(Re=100000, Pr=1.1, L=5, Di=.5)
41.366029684589265
```

ht.conv_internal.laminar_entry_Baehr_Stephan(Re, Pr, L, Di)
Calculates average internal convection Nusselt number for laminar flows in pipe during the thermal and velocity entry region according to [1] as shown in [2].

\[
\begin{align*}
    Nu_D &= \frac{3.657 \tanh(2.264G_zD^{-1/3} + 1.7G_zD^{-2/3}) + 0.0499G_zD \tanh(G_z^{-1})}{\tanh(2.432Pr^{1/6}G_zD^{-1/6})}
    
    G_z &= \frac{D}{L} Re D Pr
\end{align*}
\]

Parameters
- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- L [float] Length of pipe [m]
- Di [float] Diameter of pipe [m]

Returns
- Nu [float] Nusselt number, [-]

Notes
As L gets larger, this equation becomes the constant-temperature Nusselt number.
References

[1], [2]

Examples

```python
>>> laminar_entry_Baehr_Stephan(Re=100000, Pr=1.1, L=5, Dl=.5)
72.65402046550976
```

\texttt{ht.conv_internal.turbulent_Dittus_Boelter(Re, Pr, heating=True, revised=True)}

Calculates internal convection Nusselt number for turbulent flows in pipe according to [1], and [2], a reprint of [3].

\[ Nu = m * \frac{Re}{5} Pr^n \]

Parameters

- \( Re \) [float] Reynolds number, [-]
- \( Pr \) [float] Prandtl number, [-]
- \( \text{heating} \) [bool] Indicates if the process is heating or cooling, optional
- \( \text{revised} \) [bool] Indicates if revised coefficients should be used or not

Returns

- \( Nu \) [float] Nusselt number, [-]

Notes

The revised coefficient is \( m = 0.023 \). The original form of Dittus-Boelter has a linear coefficient of 0.0243 for heating and 0.0265 for cooling. These are sometimes rounded to 0.024 and 0.026 respectively. The default, heating, provides \( n = 0.4 \). Cooling makes \( n = 0.3 \).

0.6 Pr 160 Re_{D} 10000 L/D 10

References

[1], [2], [3]

Examples

```python
>>> turbulent_Dittus_Boelter(Re=1E5, Pr=1.2)
247.40036409449127
```

\texttt{ht.conv_internal.turbulent_Sieder_Tate(Re, Pr, mu=None, mu_w=None)}

Calculates internal convection Nusselt number for turbulent flows in pipe according to [1] and supposedly [2].

\[ Nu = 0.027 Re^{4/5} Pr^{1/3} \left( \frac{\mu}{\mu_s} \right)^{0.14} \]

Parameters

2.10. Internal convection (ht.conv_internal)
Re [float] Reynolds number, [-]
Pr [float] Prandtl number, [-]
mu [float] Viscosity of fluid, [Pa*s]
mu_w [float] Viscosity of fluid at wall temperature, [Pa*s]

Returns

Nu [float] Nusselt number, [-]

Notes

A linear coefficient of 0.023 is often listed with this equation. The source of the discrepancy is not known. The equation is not present in the original paper, but is nevertheless the source usually cited for it.

References

[1], [2]

Examples

```python
>>> turbulent_Sieder_Tate(Re=1E5, Pr=1.2)
286.9178136793052
>>> turbulent_Sieder_Tate(Re=1E5, Pr=1.2, mu=0.01, mu_w=0.067)
219.84016455766044
```

ht.conv_internal.turbulent_entry_Hausen(Re, Pr, Di, x)
Calculates internal convection Nusselt number for the entry region of a turbulent flow in pipe according to [2] as in [1].

\[ Nu = 0.037(Re^{0.75} - 180)Pr^{0.42}[1 + (x/D)^{-2/3}] \]

Parameters

Re [float] Reynolds number, [-]
Pr [float] Prandtl number, [-]
Di [float] Inside diameter of pipe, [m]
x [float] Length inside of pipe for calculation, [m]

Returns

Nu [float] Nusselt number, [-]

Notes

Range according to [1] is 0.7 < Pr < 3 and 10^4 Re < 5*10^6.

References

[1], [2]
Examples

```python
>>> turbulent_entry_Hausen(Re=1E5, Pr=1.2, Di=0.154, x=0.05)
677.7228275901755
```

ht.conv_internal.turbulent_Colburn\((Re, Pr)\)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[ Nu = 0.023 Re^{0.8} Pr^{1/3} \]

Parameters

- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]

Returns

- **Nu** [float] Nusselt number, [-]

Notes

Range according to [1] is 0.5 < Pr < 3 and 10^4 < Re < 10^5.

References

[1], [2]

Examples

```python
>>> turbulent_Colburn(Re=1E5, Pr=1.2)
244.41147091200068
```

ht.conv_internal.turbulent_Drexel_McAdams\((Re, Pr)\)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[ Nu = 0.021 Re^{0.8} Pr^{0.4} \]

Parameters

- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]

Returns

- **Nu** [float] Nusselt number, [-]

Notes

Range according to [1] is Pr 0.7 and 10^4 Re 5*10^6.

References

[1], [2]
Examples

```python
>>> turbulent_Drexel_McAdams(Re=1E5, Pr=0.6)
171.19055301724387
```

```
ht.conv_internal.turbulent_von_Karman(Re, Pr, fd)
```

Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[
Nu = \frac{(f/8)RePr}{1 + 5(f/8)^{0.5} [Pr - 1 + \ln (\frac{5Pr+1}{6})]}
\]

Parameters

- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
- **fd** [float] Darcy friction factor [-]

Returns

- **Nu** [float] Nusselt number, [-]

Notes

Range according to [1] is 0.5 Pr 3 and 10^4 Re 10^5.

References

[1], [2]

Examples

```python
>>> turbulent_von_Karman(Re=1E5, Pr=1.2, fd=0.0185)
255.7243541243272
```

```
ht.conv_internal.turbulent_Prandtl(Re, Pr, fd)
```

Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[
Nu = \frac{(f/8)RePr}{1 + 8.7(f/8)^{0.5}(Pr - 1)}
\]

Parameters

- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
- **fd** [float] Darcy friction factor [-]

Returns

- **Nu** [float] Nusselt number, [-]

Notes

Range according to [1] 0.5 Pr 5 and 10^4 Re 5*10^6
References

[1], [2]

Examples

```python
>>> turbulent_Prandtl(Re=1E5, Pr=1.2, fd=0.0185)
256.073339689557
```

ht.conv_internal.turbulent_Friend_Metzner(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[
Nu = \frac{(f/8)RePr}{1.2 + 11.8(f/8)^{0.5}(Pr - 1)Pr^{-1/3}}
\]

Parameters

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- fd [float] Darcy friction factor [-]

Returns

- Nu [float] Nusselt number, [-]

Notes

Range according to [1] 50 < Pr 600 and 5*10^4 Re 5*10^6. The extreme limits on range should be considered!

References

[1], [2]

Examples

```python
>>> turbulent_Friend_Metzner(Re=1E5, Pr=100., fd=0.0185)
1738.3356262055322
```

ht.conv_internal.turbulent_Petukhov_Kirillov_Popov(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] and [3] as in [1].

\[
Nu = \frac{(f/8)RePr}{C + 12.7(f/8)^{1/2}(Pr^{2/3} - 1)}
\]

\[
C = 1.07 + 900/Re - [0.63/(1 + 10Pr)]
\]

Parameters

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- fd [float] Darcy friction factor [-]

Returns

- Nu [float] Nusselt number, [-]
Notes

Range according to [1] is \(0.5 < Pr\ < 10^6\) and \(4000 \ < Re \ < 5 \times 10^6\)

References

[1], [2], [3]

Examples

```python
>>> turbulent_Petukhov_Kirillov_Popov(Re=1E5, Pr=1.2, fd=0.0185)
250.11935088905105
```

ht.conv_internal.turbulent_Webb(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[
Nu = \frac{(f/8)RePr}{1.07 + 9(f/8)^{0.5}(Pr - 1)Pr^{1/4}}
\]

Parameters

Re [float] Reynolds number, [-]
Pr [float] Prandtl number, [-]
fd [float] Darcy friction factor [-]

Returns

Nu [float] Nusselt number, [-]

Notes

Range according to [1] is \(0.5 < Pr\ < 100\) and \(10^4 \ < Re \ < 5 \times 10^6\)

References

[1], [2]

Examples

```python
>>> turbulent_Webb(Re=1E5, Pr=1.2, fd=0.0185)
239.10130376815872
```

ht.conv_internal.turbulent_Sandall(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1].

\[
Nu = \frac{(f/8)RePr}{12.48Pr^{2/3} - 7.853Pr^{1/3} + 3.613 \ln Pr + 5.8 + C}
\]

Parameters

Re [float] Reynolds number, [-]
Pr [float] Prandtl number, [-]
fd [float] Darcy friction factor [-]

Returns

Nu [float] Nusselt number, [-]
**Heat Transfer Documentation, Release 0.1.56**

**Prandtl number, [-]**

**Darcy friction factor [-]**

**Returns**

**Nusselt number, [-]**

**Notes**

Range according to [1] is 0.5 < Pr 2000 and 10^4 Re 5*10^6.

**References**

[1], [2]

**Examples**

```python
>>> turbulent_Sandall(Re=1E5, Pr=1.2, fd=0.0185)
229.0514352970239
```

ht.conv_internal.turbulent_Gnielinski(Re, Pr, fd)

Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1]. This is the most recent general equation, and is strongly recommended.

\[
Nu = \frac{(f/8)(Re - 1000)Pr}{1 + 12.7(f/8)^{1/2}(Pr^{2/3} - 1)}
\]

**Parameters**

- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
- **fd** [float] Darcy friction factor [-]

**Returns**

**Nusselt number, [-]**

**Notes**

Range according to [1] is 0.5 < Pr 2000 and 2300 Re 5*10^6.

**References**

[1], [2]

**Examples**

```python
>>> turbulent_Gnielinski(Re=1E5, Pr=1.2, fd=0.0185)
254.62682749359632
```
ht.conv_internal.turbulent_Gnielinski_smooth_1(Re, Pr)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1]. This is a simplified case assuming smooth pipe.

\[ Nu = 0.0214(Re^{0.8} - 100)Pr^{0.4} \]

**Parameters**

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]

**Returns**

- Nu [float] Nusselt number, [-]

**Notes**

Range according to [1] is 0.5 < Pr 1.5 and 10^4 Re 5*10^6.

**References**

[1], [2]

**Examples**

```python
>>> turbulent_Gnielinski_smooth_1(Re=1E5, Pr=1.2)
227.88800494373442
```

ht.conv_internal.turbulent_Gnielinski_smooth_2(Re, Pr)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as in [1]. This is a simplified case assuming smooth pipe.

\[ N_u = 0.012(Re^{0.87} - 280)Pr^{0.4} \]

**Parameters**

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]

**Returns**

- Nu [float] Nusselt number, [-]

**Notes**

Range according to [1] is 1.5 < Pr 500 and 3*10^3 Re 10^6.

**References**

[1], [2]
Examples

```python
>>> turbulent_Gnielinski_smooth_2(Re=1E5, Pr=7.)
577.7692524513449
```

**ht.conv_internal.turbulent_Churchill_Zajic**(*Re*, *Pr*, *fd*)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as developed in [1]. Has yet to obtain popularity.

\[
Nu = \left\{ \left( \frac{Pr_T}{Pr} \right) \frac{1}{Nu_{di}} + \left[ 1 - \left( \frac{Pr_T}{Pr} \right)^{2/3} \right] \frac{1}{Nu_{D\infty}} \right\}^{-1}
\]

\[
Nu_{di} = \frac{Re(f/8)}{1 + 145(8/f)^{-5/4}}
\]

\[
Nu_{D\infty} = 0.07343Re \left( \frac{Pr}{Pr_T} \right)^{1/3} (f/8)^{0.5}
\]

\[
Pr_T = 0.85 + \frac{0.015}{Pr}
\]

**Parameters**
- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
- **fd** [float] Darcy friction factor [-]

**Returns**
- **Nu** [float] Nusselt number, [-]

**Notes**
No restrictions on range. This is an equation that is developed with more theoretical work than others.

**References**
[1], [2]

**Examples**

```python
>>> turbulent_Churchill_Zajic(Re=1E5, Pr=1.2, fd=0.0185)
260.5564907817961
```

**ht.conv_internal.turbulent_ESDU**(*Re*, *Pr*)
Calculates internal convection Nusselt number for turbulent flows in pipe according to the ESDU as shown in [1].

\[
Nu = 0.0225 Re^{0.795} Pr^{0.495} \exp(-0.0225 \ln(Pr)^2)
\]

**Parameters**
- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
Returns

Nu [float] Nusselt number, [-]

Notes

4000 < Re < 1E6, 0.3 < Pr < 3000 and L/D > 60. This equation has not been checked. It was developed by a commercial group. This function is a small part of a much larger series of expressions accounting for many factors.

References

[1]

Examples

```python
turbulent_ESDU(Re=1E5, Pr=1.2)
```

ht.conv_internal.turbulent_Martinelli(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1].

\[ Nu = \frac{RePr(f/8)^{0.5}}{5[Pr + \ln(1 + 5Pr) + 0.5 \ln(Re(f/8)^{0.5}/60)]} \]

Parameters

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- fd [float] Darcy friction factor [-]

Returns

Nu [float] Nusselt number, [-]

Notes

No range is given for this equation. Liquid metals are probably its only applicability.

References

[1], [2]

Examples

```python
turbulent_Martinelli(Re=1E5, Pr=100., fd=0.0185)
```
ht.conv_internal.turbulent_Nunner(Re, Pr, fd, fd_smooth)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1].

\[ Nu = \frac{Re Pr (f / 8)}{1 + 1.5 Re^{-1/8} Pr^{-1/6} (Pr f / fd) - 1} \]

**Parameters**
- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
- **fd** [float] Darcy friction factor [-]
- **fd_smooth** [float] Darcy friction factor of a smooth pipe [-]

**Returns**
- **Nu** [float] Nusselt number, [-]

**Notes**
Valid for Pr 0.7; bad results for Pr > 1.

**References**
[1], [2]

**Examples**

```python
>>> turbulent_Nunner(Re=1E5, Pr=0.7, fd=0.0185, fd_smooth=0.005)
101.15841010919947
```

ht.conv_internal.turbulent_Dipprey_Sabersky(Re, Pr, fd, eD)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1].

\[ Nu = \frac{Re Pr (f / 8)}{1 + (f / 8)^{0.5} [5.19 Re^{0.2} Pr^{0.44} - 8.48]} \]

**Parameters**
- **Re** [float] Reynolds number, [-]
- **Pr** [float] Prandtl number, [-]
- **fd** [float] Darcy friction factor [-]
- **eD** [float] Relative roughness, [-]

**Returns**
- **Nu** [float] Nusselt number, [-]

**Notes**
According to [1], the limits are: 1.2 Pr 5.94 and 1.4*10^4 Re 5E5 and 0.0024 eD 0.049.
References

[1], [2]

Examples

```python
>>> turbulent_Dipprey_Sabersky(Re=1E5, Pr=1.2, fd=0.0185, eD=1E-3)
288.33365198566656
```

ht.conv_internal.turbulent_Gowen_Smith(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1].

\[
Nu = \frac{RePr(f/8)^{0.5}}{4.5 + 0.155(Re(f/8)^{0.5})^{0.54} + (8/f)^{0.5}Pr^{0.5}}
\]

Parameters

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- fd [float] Darcy friction factor [-]

Returns

- Nu [float] Nusselt number, [-]

Notes

0.7 Pr 14.3 and 10^4 Re 5E4 and 0.0021 eD 0.095

References

[1], [2]

Examples

```python
>>> turbulent_Gowen_Smith(Re=1E5, Pr=1.2, fd=0.0185)
131.72530453824106
```

ht.conv_internal.turbulent_Kawase_Ulbrecht(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1].

\[
Nu = 0.0523RePr^{0.5}(f/4)^{0.5}
\]

Parameters

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- fd [float] Darcy friction factor [-]

Returns

- Nu [float] Nusselt number, [-]
Notes

No limits are provided.

References

[1], [2]

Examples

```python
>>> turbulent_Kawase_Ulbrecht(Re=1E5, Pr=1.2, fd=0.0185)
389.6262247333975
```

ht.conv_internal.turbulent_Kawase_De(Re, Pr, fd)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1].

\[
Nu = 0.0471RePr^{0.5}(f/4)^{0.5}(1.11 + 0.44Pr^{-1/3} - 0.7Pr^{-1/6})
\]

Parameters

- Re [float] Reynolds number, [-]
- Pr [float] Prandtl number, [-]
- fd [float] Darcy friction factor [-]

Returns

- Nu [float] Nusselt number, [-]

Notes

5.1 Pr 390 and 5000 Re 5E5 and 0.0024 eD 0.165.

References

[1], [2]

Examples

```python
>>> turbulent_Kawase_De(Re=1E5, Pr=1.2, fd=0.0185)
296.5019733271324
```

ht.conv_internal.turbulent_Bhatti_Shah(Re, Pr, fd, eD)
Calculates internal convection Nusselt number for turbulent flows in pipe according to [2] as shown in [1]. The most widely used rough pipe turbulent correlation.

\[
Nu_D = \frac{(f/8)Re_DPr}{1 + \sqrt{f/8}(4.5Re_D^{0.2}Pr^{0.5} - 8.48)}
\]

Parameters

- Re [float] Reynolds number, [-]
Heat Transfer Documentation, Release 0.1.56

Pr [float] Prandtl number, [-]

fd [float] Darcy friction factor [-]
eD [float] Relative roughness, [-]

Returns

Nu [float] Nusselt number, [-]

Notes

According to [1], the limits are: 0.5 Pr 10 0.002 e/D 0.05 10,000 Re_D Another correlation is listed in this equation, with a wider variety of validity.

References

[1], [2]

Examples

```python
>>> turbulent_Bhatti_Shah(Re=1E5, Pr=1.2, fd=0.0185, eD=1E-3)
302.7037617414273
```

ht.conv_internal.Nu_conv_internal(Re, Pr, eD=0.0, Di=None, x=None, fd=None, Method=None)

This function calculates the heat transfer coefficient for internal convection inside a circular pipe.

Requires at a minimum a flow’s Reynolds and Prandtl numbers Re and Pr. Relative roughness eD can be specified to include the enhancement of heat transfer from the added turbulence.

For laminar flow, thermally and hydraulically developing flow is supported with the pipe diameter Di and distance x is provided.

If no correlation’s name is provided as Method, the most accurate applicable correlation is selected.

- If laminar, x and Di provided: ‘Baehr-Stephan laminar thermal/velocity entry’
- Otherwise if laminar, no entry information provided: ‘Laminar - constant T’ (Nu = 3.66)
- If turbulent and Pr < 0.03: ‘Martinelli’
- If turbulent, x and Di provided: ‘Hausen’
- Otherwise if turbulent: ‘Churchill-Zajic’

Parameters

Re [float] Reynolds number, [-]

Pr [float] Prandtl number, [-]
eD [float, optional] Relative roughness, [-]

Di [float, optional] Inside diameter of pipe, [m]

x [float, optional] Length inside of pipe for calculation, [m]

fd [float, optional] Darcy friction factor [-]

Returns
Nu  [float] Nusselt number, [-]

Other Parameters

Method  [string, optional] A string of the function name to use, as in the dictionary vertical_cylinder_correlations

Examples

Turbulent example

```python
>>> Nu_conv_internal(Re=1E5, Pr=.7)
183.71057902604906
```

Entry length - laminar example

```python
>>> Nu_conv_internal(Re=1E2, Pr=.7, x=.01, Di=.1)
14.91799128769779
```

ht.conv_internal.Nu_conv_internal_methods(Re, Pr, eD=0, Di=None, x=None, fd=None, check_ranges=True)

This function returns a list of correlation names for the calculation of heat transfer coefficient for internal convection inside a circular pipe.

Parameters

- Re  [float] Reynolds number, [-]
- Pr  [float] Prandtl number, [-]
- eD  [float, optional] Relative roughness, [-]
- Di  [float, optional] Inside diameter of pipe, [m]
- x   [float, optional] Length inside of pipe for calculation, [m]
- fd  [float, optional] Darcy friction factor [-]
- check_ranges  [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

- methods  [list] List of methods which can be used to calculate Nu with the given inputs

Examples

Turbulent example

```python
>>> Nu_conv_internal_methods(Re=1E5, Pr=.7)[0]
'Churchill-Zajic'
```

Entry length - laminar example

```python
>>> Nu_conv_internal_methods(Re=1E2, Pr=.7, x=.01, Di=.1)[0]
'Baehr-Stephan laminar thermal/velocity entry'
```
ht.conv_internal.Morimoto_Hotta(Re, Pr, Dh, Rm)
Calculates Nusselt number for flow inside a spiral heat exchanger of spiral mean diameter Rm and hydraulic
diameter Dh according to [1], also as shown in [2] and [3].

\[ Nu = 0.0239 \left( 1 + 5.54 \frac{D_h}{R_m} \right) Re^{0.806} Pr^{0.268} \]

\[ D_h = \frac{2HS}{H+S} \]

\[ R_m = \frac{R_{\text{min}} + R_{\text{max}}}{2} \]

Parameters

Re [float] Reynolds number with bulk properties, [-]
Pr [float] Prandtl number with bulk properties [-]
Dh [float] Average hydraulic diameter, [m]
Rm [float] Average spiral radius, [m]

Returns

Nu [float] Nusselt number with respect to Dh, [-]

Notes


References

[1], [2], [3]

Examples

```python
>>> Morimoto_Hotta(1E5, 5.7, .05, .5)
634.4879473869859
```

ht.conv_internal.helical_turbulent_Nu_Mori_Nakayama(Re, Pr, Di, Dc)
Calculates Nusselt number for a fluid flowing inside a curved pipe such as a helical coil under turbulent condi-
tions, using the method of Mori and Nakayama [1], also shown in [2] and [3].

For \( Pr < 1 \):

\[ Nu = \frac{Pr}{26.2(Pr^{2/3} - 0.074)} Re^{0.8} \left( \frac{D_i}{D_c} \right)^{0.1} \left[ 1 + \frac{0.098}{Re \left( \frac{D_i}{D_c} \right)^2} \right]^{0.2} \]

For \( Pr \geq 1 \):

\[ Nu = \frac{Pr^{0.4}}{41} Re^{5/6} \left( \frac{D_i}{D_c} \right)^{1/12} \left[ 1 + \frac{0.061}{Re \left( \frac{D_i}{D_c} \right)^{2.5/6}} \right]^{1/6} \]
Parameters

- **Re** [float] Reynolds number with \( D=Di \), [-]
- **Pr** [float] Prandtl number with bulk properties [-]
- **Di** [float] Inner diameter of the coil, [m]
- **Dc** [float] Diameter of the helix/coil measured from the center of the tube on one side to the center of the tube on the other side, [m]

Returns

- **Nu** [float] Nusselt number with respect to \( Di \), [-]

Notes

At very low curvatures, the predicted heat transfer coefficient grows unbounded.

Applicable for \( Re \left( \frac{D_i}{D_c} \right)^2 > 0.1 \)

References

[1], [2], [3]

Examples

```python
>>> helical_turbulent_Nu_Mori_Nakayama(2E5, 0.7, 0.01, .2)
496.2522480663327
```

Calculates Nusselt number for a fluid flowing inside a curved pipe such as a helical coil under turbulent conditions, using the method of Schmidt [1], also shown in [2], [3], and [4].

For \( Re_{crit} < Re < 2.2 \times 10^4 \):

\[
Nu = 0.023 \left[ 1 + 14.8 \left( 1 + \frac{D_i}{D_c} \right) \left( \frac{D_i}{D_c} \right)^{1/3} \right] Re^{0.8 - 0.22 \left( \frac{D_i}{D_c} \right)^{0.1}} Pr^{1/3}
\]

For \( 2.2 \times 10^4 < Re < 1.5 \times 10^5 \):

\[
Nu = 0.023 \left[ 1 + 3.6 \left( 1 - \frac{D_i}{D_c} \right) \left( \frac{D_i}{D_c} \right)^{0.8} \right] Re^{0.8} Pr^{1/3}
\]

Parameters

- **Re** [float] Reynolds number with \( D=Di \), [-]
- **Pr** [float] Prandtl number with bulk properties [-]
- **Di** [float] Inner diameter of the coil, [m]
- **Dc** [float] Diameter of the helix/coil measured from the center of the tube on one side to the center of the tube on the other side, [m]

Returns

- **Nu** [float] Nusselt number with respect to \( Di \), [-]
Notes

For very low curvatures, reasonable results are returned by both cases of Reynolds numbers.

References

[1], [2], [3], [4]

Examples

```python
>>> helical_turbulent_Nu_Schmidt(2E5, 0.7, 0.01, .2)
466.2569996832083
```

Calculates Nusselt number for a fluid flowing inside a curved pipe such as a helical coil under turbulent conditions, using the method of Xin and Ebadian [1], also shown in [2] and [3].

For \(Re_{crit} < Re < 1 \times 10^5\):

\[
Nu = 0.00619Re^{0.92}Pr^{0.4} \left[1 + 3.455 \left(\frac{D_i}{D_c}\right)\right]
\]

Parameters

- **Re** [float] Reynolds number with \(D=D_i\), [-]
- **Pr** [float] Prandtl number with bulk properties [-]
- **Di** [float] Inner diameter of the coil, [m]
- **Dc** [float] Diameter of the helix/coil measured from the center of the tube on one side to the center of the tube on the other side, [m]

Returns

- **Nu** [float] Nusselt number with respect to \(D_i\), [-]

Notes

For very low curvatures, reasonable results are returned.

The correlation was developed with data in the range of \(0.7 < Pr < 5; 0.0267 < \frac{D_i}{D_c} < 0.0884\).

References

[1], [2], [3]

Examples

```python
>>> helical_turbulent_Nu_Xin_Ebadian(2E5, 0.7, 0.01, .2)
474.11413424344755
```
ht.conv_internal.Nu_laminar_rectangular_Shan_London(a_r)
Calculates internal convection Nusselt number for laminar flows in a rectangular pipe of varying aspect ratio, as developed in [1].

This model is derived assuming a constant wall heat flux from all sides. This is entirely theoretically derived and reproduced experimentally.

\[
Nu_{lam} = 8.235 \left( 1 - 2.0421\alpha + 3.0853\alpha^2 - 2.4765\alpha^3 + 1.0578\alpha^4 - 0.1861\alpha^5 \right)
\]

**Parameters**

- a_r [float] The aspect ratio of the channel, from 0 to 1 [-]

**Returns**

Nu [float] Nusselt number of flow in a rectangular channel, [-]

**Notes**

At an aspect ratio of 1 (square channel), the Nusselt number converges to 3.610224. The authors of [1] also published [2], which tabulates in their table 42 some less precise results that are used to check this function.

**References**

[1], [2]

**Examples**

```python
>>> Nu_laminar_rectangular_Shan_London(.7)
3.751762675455
```

### 2.11 Convection to jacketed vessels (ht.conv_jacket)

ht.conv_jacket.Lehrer(m, Dtank, Djacket, H, Dinlet, rho, Cp, k, mu, muw=None, isobaric_expansion=None, dT=None, inlettype='tangential', inletlocation='auto')
Calculates average heat transfer coefficient for a jacket around a vessel according to [1] as described in [2].

\[
Nu_{S,L} = \left[ \frac{0.03Re_S^{0.75}Pr}{1 + 1.74(Pr-1)} \right] \left( \frac{\mu}{\mu_w} \right)^{0.14} \\
d_g = \left( \frac{8}{3} \right)^{0.5} \delta \\
v_h = \left( \frac{v_Sv_{inlet}}{1} \right)^{0.5} + v_A \\
v_{inlet} = \frac{Q}{\pi d_{inlet}^2} \\
v_s = \frac{Q}{\frac{\pi}{4}(D_{jacket}^2 - D_{tank}^2)}
\]

2.11. Convection to jacketed vessels (ht.conv_jacket)
For Radial inlets:

\[ v_A = 0.5(2gH\beta\Delta T)^{0.5} \]

For Tangential inlets:

\[ v_A = 0 \]

**Parameters**

- \( m \) [float] Mass flow rate of fluid, [kg/s]
- \( D_{tank} \) [float] Outer diameter of tank or vessel surrounded by jacket, [m]
- \( D_{jacket} \) [float] Inner diameter of jacket surrounding a vessel or tank, [m]
- \( H \) [float] Height of the vessel or tank, [m]
- \( D_{inlet} \) [float] Inner diameter of inlet into the jacket, [m]
- \( \rho \) [float] Density of the fluid at \( T_m \) [kg/m\(^3\)]
- \( C_p \) [float] Heat capacity of fluid at \( T_m \) [J/kg/K]
- \( k \) [float] Thermal conductivity of fluid at \( T_m \) [W/m/K]
- \( \mu \) [float] Viscosity of fluid at \( T_m \) [Pa*s]
- \( \mu_w \) [float, optional] Viscosity of fluid at \( T_w \) [Pa*s]
- \( isobaric\_expansion \) [float, optional] Constant pressure expansivity of a fluid, [m\(^3\)/mol/K]
- \( dT \) [float, optional] Temperature difference of fluid in jacket, [K]
- \( inlettype \) [str, optional] Either ‘tangential’ or ‘radial’
- \( inletlocation \) [str, optional] Either ‘top’ or ‘bottom’ or ‘auto’

**Returns**

- \( h \) [float] Average heat transfer coefficient inside the jacket [W/m\(^2\)/K]

**Notes**

If the fluid is heated and enters from the bottom, natural convection assists the heat transfer and the Grashof term is added; if it were to enter from the top, it would be subtracted. The situation is reversed if entry is from the top.

**References**

[1], [2]

**Examples**

Example as in [2], matches completely.

```python
>>> Lehrer(m=2.5, Dtank=0.6, Djacket=0.65, H=0.6, Dinlet=0.025, dT=20.,
... rho=995.7, Cp=4178.1, k=0.615, mu=798E-6, muw=355E-6)
2922.128124761829
```

Examples similar to in [2] but covering the other case:
ht.conv_jacket.Stein_Schmidt \( m, D_{tank}, D_{jacket}, H, D_{inlet}, \rho, C_p, k, \mu, \mu_w=\text{None}, \rho_w=\text{None}, \text{inlettype='tangential'}, \text{inletlocation='auto'}, \text{roughness=0.0} \)

Calculates average heat transfer coefficient for a jacket around a vessel according to [1] as described in [2].

\[
l_{ch} = \left( \frac{\pi}{2} \right)^2 \frac{D_{tank}^2 + H^2}{0.5}
\]

\[
d_{ch} = 2\delta
\]

\[
Re_j = \frac{v_{ch}d_{ch}\rho}{\mu}
\]

\[
Gr_J = \frac{g\rho(\rho - \rho_w)d_{ch}^3}{\mu^2}
\]

\[
Re_{J,eq} = \left[ Re_j^2 \pm \left( \frac{|Gr_J|\frac{H}{d_{ch}}}{50} \right) \right]^{0.5}
\]

\[
Nu_J = (Nu_A^3 + Nu_B^3 + Nu_C^3 + Nu_D^3)^{1/3}\left( \frac{\mu}{\mu_w} \right)^{0.14}
\]

\[
Nu_J = \frac{hd_{ch}}{k}
\]

\[
Nu_A = 3.66
\]

\[
Nu_B = 1.62Pr^{1/3}Re_{J,eq}^{1/3}\left( \frac{d_{ch}}{l_{ch}} \right)^{1/3}
\]

\[
Nu_C = 0.664Pr^{1/3}(Re_{J,eq})^{0.5}
\]

\[
\text{if} \ Re_{J,eq} < 2300: \ Nu_D = 0
\]

\[
Nu_D = 0.0115Pr^{1/3}Re_{J,eq}^{0.9}\left( 1 - \left( \frac{2300}{Re_{J,eq}} \right)^{2.5} \right)\left( 1 + \left( \frac{d_{ch}}{l_{ch}} \right)^{2/3} \right)
\]

For Radial inlets:

\[
v_{ch} = v_{Mit} = \frac{\ln \left( b_{Mit} \right)}{1 - \frac{b_{E_in}}{b_{Mit}}}
\]

\[
b_{E_in} = \frac{\pi D_{inlet}^2}{8}\delta
\]

\[
b_{Mit} = \frac{\pi}{2}D_{tank}\sqrt{1 + \frac{\pi^2 D_{tank}^4}{4H^2}}
\]

\[
v_{Mit} = \frac{Q}{2\delta b_{Mit}}
\]

For Tangential inlets:

\[
v_{ch} = \left( v_x^2 + v_z^2 \right)^{0.5}
\]

2.11. Convection to jacketed vessels (ht.conv_jacket)
\[ v_x = v_{inlet} \left( \ln \left[ 1 + \frac{f_d D_{tank} H}{D_{inlet}^2} \frac{v_x(0)}{v_{inlet}} \right] \right) \]

\[ v_x(0) = K_3 + (K_3^2 + K_4)^{0.5} \]

\[ K_3 = \frac{v_{inlet}}{4} - \frac{D_{inlet}^2 v_{inlet}}{4 f_d D_{tank} H} \]

\[ K_4 = \frac{D_{inlet}^2 v_{inlet}^2}{2 f_d D_{tank} H} \]

\[ v_z = \frac{Q}{\pi D_{tank} \delta} \]

\[ v_{inlet} = \frac{Q}{\frac{1}{4} D_{inlet}^2} \]

**Parameters**

- **m** [float] Mass flow rate of fluid, [kg/m^3]
- **Dtank** [float] Outer diameter of tank or vessel surrounded by jacket, [m]
- **Djacket** [float] Inner diameter of jacket surrounding a vessel or tank, [m]
- **H** [float] Height of the vessel or tank, [m]
- **Dinlet** [float] Inner diameter of inlet into the jacket, [m]
- **rho** [float] Density of the fluid at Tm [kg/m^3]
- **Cp** [float] Heat capacity of fluid at Tm [J/kg/K]
- **k** [float] Thermal conductivity of fluid at Tm [W/m/K]
- **mu** [float] Viscosity of fluid at Tm [Pa*s]
- **muw** [float, optional] Viscosity of fluid at Tw [Pa*s]
- **rhow** [float, optional] Density of the fluid at Tw [kg/m^3]
- **inlettype** [str, optional] Either ‘tangential’ or ‘radial’
- **inletlocation** [str, optional] Either ‘top’ or ‘bottom’ or ‘auto’
- **roughness** [float, optional] Roughness of the tank walls [m]

**Returns**

- **h** [float] Average transfer coefficient inside the jacket [W/m^2/K]

**Notes**

[1] is in German and has not been reviewed. Multiple other formulations are considered in [1].

If the fluid is heated and enters from the bottom, natural convection assists the heat transfer and the Grashof term is added; if it were to enter from the top, it would be subtracted. The situation is reversed if entry is from the top.

**References**

[1], [2]
Examples

Example as in [2], matches in all but friction factor:

```python
>>> Stein_Schmidt (m=2.5, Dtank=0.6, Djacket=0.65, H=0.6, Dinlet=0.025,
... rho=995.7, Cp=4178.1, k=0.615, mu=798E-6, muw=355E-6, rhow=971.8)
5695.2041698088615
```

## 2.12 Convection to packed beds (ht.conv_packed_bed)

`ht.conv_packed_bed.Nu_packed_bed_Gnielinski` (*dp*, *voidage*, *vs*, *rho*, *mu*, *Pr*, *fa=None*)

Calculates Nusselt number of a fluid passing over a bed of particles using a correlation shown in [3] and cited as from [1] and [2]. Likely the best available model as the author of [1] is the same as [2] and [3].

\[
Nu = f_a Nu_{sphere}
\]

\[
Nu_{sphere} = 2 + \sqrt{Nu_{m, lam}^2 + Nu_{m, turb}^2}
\]

\[
Nu_{m, lam} = 0.664 Re^{0.5} Pr^{1/3}
\]

\[
Nu_{m, turb} = \frac{0.037 Re^{0.8} Pr}{1 + 2.443 Re^{-0.1} (Pr^{2/3} - 1)}
\]

\[
Re = \frac{\rho v_s d_p}{\mu}
\]

**Parameters**

- **dp** [float] Equivalent spherical particle diameter of packing [m]
- **voidage** [float] Void fraction of bed packing [-]
- **vs** [float] Superficial velocity of the fluid [m/s]
- **rho** [float] Density of the fluid [kg/m^3]
- **mu** [float] Viscosity of the fluid [Pa*s]
- **Pr** [float] Prandtl number of the fluid []
- **fa** [float, optional] Fator increasing heat transfer []

**Returns**

- **Nu** [float] Nusselt number for heat transfer to the packed bed [-]

**Notes**

*fa* is a factor relating how much more heat transfer happens than would normally, around one sphere. For spheres of the same size, \( f_a = 1 + 1.5(1 - \epsilon) \). For cylinders with l/d ratio of 0.24 < l/d < 1.2 use \( fa = 1.6 \). For cubes, use \( fa = 1.6 \) For Raschig rings, use \( fa = 2.1 \) For Berl saddles, use \( fa = 2.3 \). \( fa \) is calculated with the relationship for spheres if not provided.

Confirmed with experimental data for a range of \( 1E - 1 < Re < 1,000 \) and \( 0.4 < Pr < 1000 \) for spheres. Limits are smaller for other shapes.
References

[1], [2], [3]

Examples

```python
>>> Nu_packed_bed_Gnielinski(dp=8E-4, voidage=0.4, vs=1, rho=1E3, mu=1E-3, Pr=0.7)
61.37823202546954
```

ht.conv_packed_bed.Nu_Wakao_Kagei(Re, Pr)
Calculates Nusselt number of a fluid passing over a bed of particles using a correlation shown in [1] and also cited in the review of [2]. Relatively rough, as it has no dependence on voidage.

\[
Nu = 2 + 1.1Pr^{1/3}Re^{0.6}
\]

Parameters

Re [float] Reynolds number with pebble diameter as characteristic dimension, [-]
Pr [float] Prandtl number of the fluid []

Returns

Nu [float] Nusselt number for heat transfer to the packed bed [-]

Notes

Fit for Re from 3 to 3000; claimed reasonableness of fit to to 1E6.

References

[1], [2]

Examples

```python
>>> Nu_Wakao_Kagei(2000, 0.7)
95.40641328041248
```

ht.conv_packed_bed.Nu_Achenbach(Re, Pr, voidage)
Calculates Nusselt number of a fluid passing over a bed of particles using a correlation shown in [1] and also cited in the review of [2].

\[
Nu = [(1.18Re^{0.58})^4 + (0.23\left(\frac{Re}{1-\epsilon}\right)^{0.75})^4]^{0.25}
\]

Parameters

Re [float] Reynolds number with pebble diameter as characteristic dimension, [-]
Pr [float] Prandtl number of the fluid []
voidage [float] Void fraction of bed packing [-]

Returns

Nu [float] Nusselt number for heat transfer to the packed bed [-]
Notes

Claimed value for Re/\epsilon < 7.7E5 Developed with tests performed in a wind tunnel at conditions up to 30 bar.

References

[1], [2]

Examples

```python
>>> Nu_Achenbach(2000, 0.7, 0.4)
117.70343608599121
```

ht.conv_packed_bed.Nu_KTA(Re, Pr, voidage)

Calculates Nusselt number of a fluid passing over a bed of particles using a correlation shown in [1] and also cited in the review of [2].

\[
N_u = 1.27 \frac{Pr^{1/3}}{\epsilon^{1/18}} Re^{0.36} + 0.033 \frac{Pr^{0.5}}{\epsilon^{1/7}} Re^{0.86}
\]

Parameters

- Re [float] Reynolds number with pebble diameter as characteristic dimension, [-]
- Pr [float] Prandtl number of the fluid [-]
- voidage [float] Void fraction of bed packing [-]

Returns

- Nu [float] Nusselt number for heat transfer to the packed bed [-]

Notes

100 < Re < 1E5; 0.36 < \epsilon < 0.42; D/d > 20 with D as bed diameter, d as particle diameter; H > 4d with H as bed height.

References

[1], [2]

Examples

```python
>>> Nu_KTA(2000, 0.7, 0.4)
102.08516480718129
```
2.13 Convection to Plate Heat Exchangers (single-phase) (ht.conv_plate)

ht.conv_plate.Nu_plate_Kumar(Re, Pr, chevron_angle, mu=None, mu_wall=None)
Calculates Nusselt number for single-phase flow in a well-designed Chevron-style plate heat exchanger according to [1]. The data is believed to have been developed by APV International Limited, since acquired by SPX Corporation. This uses a curve fit of that data published in [2].

\[ Nu = C_1 Re^m Pr^{0.33} \left( \frac{\mu}{\mu_{wall}} \right)^{0.17} \]

\( C_1 \) and \( m \) are coefficients looked up in a table, with varying ranges of \( Re \) validity and chevron angle validity. See the source for their exact values. The wall fluid property correction is included only if the viscosity values are provided.

Parameters

- **Re** [float] Reynolds number with respect to the hydraulic diameter of the channels, [-]
- **Pr** [float] Prandtl number calculated with bulk fluid properties, [-]
- **chevron_angle** [float] Angle of the plate corrugations with respect to the vertical axis (the direction of flow if the plates were straight), between 0 and 90. Many plate exchangers use two alternating patterns; use their average angle for that situation [degrees]
- **mu** [float, optional] Viscosity of the fluid at the bulk (inlet and outlet average) temperature, [Pa*s]
- **mu_wall** [float, optional] Viscosity of fluid at wall temperature, [Pa*s]

Returns

- **Nu** [float] Nusselt number with respect to \( Dh \), [-]

Notes

Data on graph from \( Re=0.1 \) to \( Re=10000 \), with chevron angles 30 to 65 degrees. See PlateExchanger for further clarification on the definitions.

It is believed the constants used in this correlation were curve-fit to the actual graph in [1] by the author of [2] as there is no

As the coefficients change, there are numerous small discontinuities, although the data on the graphs is continuous with sharp transitions of the slope.

The author of [1] states clearly this correlation is “applicable only to well designed Chevron PHEs”.

References

[1], [2]

Examples

```python
>>> Nu_plate_Kumar(Re=2000, Pr=0.7, chevron_angle=30)
47.757818892853955
```

With the wall-correction factor included:
ht.conv_plate.Nu_plate_Martin(Re, Pr, plate_enlargement_factor, variant='1999')
Calculates Nusselt number for single-phase flow in a Chevron-style plate heat exchanger according to [1], also shown in [2] and [3].

\[ Nu = 0.122Pr^{1/3} \left[ f_d Re^2 \sin(2\phi) \right]^{0.374} \]

The Darcy friction factor should be calculated with the Martin (1999) friction factor correlation, as that is what the power of 0.374 was regressed with. It can be altered to a later formulation by Martin in the VDI Heat Atlas 2E, which increases the calculated heat transfer friction slightly.

Parameters

- **Re** [float] Reynolds number with respect to the hydraulic diameter of the channels, [-]
- **Pr** [float] Prandtl number calculated with bulk fluid properties, [-]
- **plate_enlargement_factor** [float] The extra surface area multiplier as compared to a flat plate caused the corrugations, [-]
- **variant** [str] One of ‘1999’ or ‘VDI’; chooses between the two Martin friction factor correlations, [-]

Returns

- **Nu** [float] Nusselt number with respect to \( Dh \), [-]

Notes

Based on experimental data from Re from 200 - 10000 and enhancement factors calculated with chevron angles of 0 to 80 degrees. See PlateExchanger for further clarification on the definitions.

Note there is a discontinuity at Re = 2000 for the transition from laminar to turbulent flow, arising from the friction factor correlation’s transition ONLY, although the literature suggests the transition is actually smooth.

A viscosity correction power for liquid flows of (1/6) is suggested, and for gases, no correction factor.

References

[1], [2], [3]

Examples

```python
>>> Nu_plate_Martin(Re=2000, Pr=.7, plate_enlargement_factor=1.18)
43.5794551998615
```

ht.conv_plate.Nu_plate_Muley_Manglik(Re, Pr, chevron_angle, plate_enlargement_factor)
Calculates Nusselt number for single-phase flow in a Chevron-style plate heat exchanger according to [1], also shown in [2] and [3].

\[ Nu = [0.2668 - 0.006967(\beta) + 7.244 \times 10^{-5}(\beta)^2] \times [20.7803 - 50.9372\phi + 41.1585\phi^2 - 10.1507\phi^3] \times Re^{0.728+0.0543\sin(2\phi)} \]

Parameters

- **Re** [float] Reynolds number with respect to the hydraulic diameter of the channels, [-]
**Heat Transfer Documentation, Release 0.1.56**

**Pr** [float] Prandtl number calculated with bulk fluid properties, [-]

**chevron_angle** [float] Angle of the plate corrugations with respect to the vertical axis (the direction of flow if the plates were straight), between 0 and 90. Many plate exchangers use two alternating patterns; use their average angle for that situation [degrees]

**plate_enlargement_factor** [float] The extra surface area multiplier as compared to a flat plate caused the corrugations, [-]

**Returns**

**Nu** [float] Nusselt number with respect to $D_h$, [-]

**Notes**

The correlation as presented in [1] suffers from a typo, with a coefficient of 10.51 instead of 10.15. Several more decimal places were published along with the corrected typo in [2]. This has a very large difference if not implemented.

The viscosity correction power is recommended to be the blanket Sieder and Tate (1936) value of 0.14.

The correlation is recommended in the range of Reynolds numbers above 1000, chevron angles between 30 and 60 degrees, and enlargement factors from 1 to 1.5. Due to its cubic nature it is not likely to give good results if the chevron angle or enlargement factors are out of those ranges.

**References**

[1],[2]

**Examples**

```python
>>> Nu_plate_Muley_Manglik(Re=2000, Pr=.7, chevron_angle=45, 
... plate_enlargement_factor=1.18)
36.49087100602062
```

ht.conv_plate.Nu_plate_Khan_Khan($Re, Pr, chevron\_angle$)

Calculates Nusselt number for single-phase flow in a Chevron-style plate heat exchanger according to [1].

$$Nu = \left(0.0161 \frac{\beta}{\beta_{max}} + 0.1298 \right) Re^{(0.198 \frac{\beta}{\beta_{max}} + 0.6398)} Pr^{0.35}$$

**Parameters**

**Re** [float] Reynolds number with respect to the hydraulic diameter of the channels, [-]

**Pr** [float] Prandtl number calculated with bulk fluid properties, [-]

**chevron_angle** [float] Angle of the plate corrugations with respect to the vertical axis (the direction of flow if the plates were straight), between 0 and 90. Many plate exchangers use two alternating patterns; use their average angle for that situation [degrees]

**Returns**

**Nu** [float] Nusselt number with respect to $D_h$, [-]
Notes

The viscosity correction power is recommended to be the blanket Sieder and Tate (1936) value of 0.14. The correlation is recommended in the range of Reynolds numbers from 500 to 2500, chevron angles between 30 and 60 degrees, and Prandtl numbers between 3.5 and 6.

References

[1]

Examples

```python
>>> Nu_plate_Khan_Khan(Re=1000, Pr=4.5, chevron_angle=30)
38.40883639103741
```

## 2.14 Convection with supercritical fluids (ht.conv_supercritical)

ht.conv_supercritical.Nu_McAdams(Re, Pr)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

Found in [2] to fit the enhanced heat transfer regime with a MAD of 10.3% which was better than and of the other reviewed correlations.

\[
Nu_b = 0.0243Re_b^{0.8}Pr_b^{0.4}
\]

Parameters

- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr** [float] Prandtl number with bulk fluid properties, [-]

Returns

- **Nu** [float] Nusselt number with bulk fluid properties, [-]

Notes

This has also been one of the forms of the Dittus-Boelter correlations. Claimed to fit data for high pressures and low heat fluxes.

References

[1], [2]

Examples

```python
>>> Nu_McAdams(1E5, 1.2)
261.3838629346147
```
ht.conv_supercritical.\texttt{Nu\_Shitsman}(Re, Pr\_b, Pr\_w)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1] and [2] as shown in both [3] and [4].

\[ Nu_b = 0.023 Re_b^{0.8} (\min(Pr_b, Pr_w))^{0.8} \]

### Parameters
- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr\_b** [float] Prandtl number with bulk fluid properties, [-]
- **Pr\_w** [float] Prandtl number with wall fluid properties, [-]

### Returns
- **Nu** [float] Nusselt number with bulk fluid properties, [-]

### Notes
[3] states this correlation was developed with \( D = 7.8 \) and \( 8.2 \) mm and with a \( Pr \) approximately 1. [3] ranked it third in the enhanced heat transfer category, with a MAD as 11.5%.

[4] cites [1] as the source of the correlation. Neither have been reviewed, and both are in Russian. [4] lists this as third most accurate of the 14 correlations reviewed from a database of all regimes.

### References
[1], [2], [3], [4]

### Examples

```python
>>> Nu_Shitsman(1E5, 1.2, 1.6)
266.1171311047253
```

ht.conv_supercritical.\texttt{Nu\_Griem}(Re, Pr, H=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1], also shown in [2], [3] and [4]. Has complicated rules regarding where properties should be evaluated.

\[ Nu_m = 0.0169 Re_b^{0.8356} Pr_s^{0.432} \]

### Parameters
- **Re** [float] Reynolds number as explained below, [-]
- **Pr** [float] Prandtl number as explained below, [-]
- **H** [float, optional] Enthalpy of water (if the fluid is water), [J/kg]

### Returns
- **Nu** [float] Nusselt number as explained below, [-]
Notes

w is calculated as follows, for water only, with a reference point from the 1967-IFC formulation. It is set to 1 if H is not provided: if Hb < 1.54E6 J/kg, w = 0.82; if Hb > 1.74E6 J/kg, w = 1; otherwise w = 0.82 + 9E-7*(Hb-1.54E6).

To determine heat capacity to be used, Cp should be calculated at 5 points, and the lowest three points should be averaged. The five points are: Tw, (Tw+Tf)/2, Tf, (Tb+Tf)/2, Tb.

Viscosity should be the bulk viscosity. Thermal conductivity should be the average of the bulk and wall values. Density should be the bulk density.

[2] states this correlation was developed with D = 10, 14, and 20 mm, P from 22 to 27 MPa, G from 300 to 2500 kg/m^2/s, and q from 200 to 700 kW/m^2. It was ranked 6th among the 14 correlations reviewed for enhanced heat transfer, with a MAD of 13.8%, and 6th overall for the three heat transfer conditions with a overall MAD of 14.8%. [3] ranked it 8th of 14 correlations for the three heat transfer conditions.


References

[1], [2], [3], [4]

Examples

```python
>>> Nu_Griem(1E5, 1.2)
275.4818576600527

ht.conv_supercritical.Nu_Jackson(Re, Pr, rho_w=None, rho_b=None, Cp_avg=None, Cp_b=None, T_b=None, T_w=None, T_pc=None)
```

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
Nu_b = 0.0183Re_b^{0.82}Pr^{0.5}\left(\frac{\rho_w}{\rho_b}\right)^{0.3}\left(\frac{\bar{C}_p}{C_{p,b}}\right)^{n}
\]

\[n = 0.4\] for \[T_b < T_w < T_{pc}\] or \[1.2T_{pc} < T_b < T_w\]

\[n = 0.4 + 0.2(T_w/T_{pc} - 1)\] for \[T_b < T_{pc} < T_w\]

\[n = 0.4 + 0.2(T_w/T_{pc} - 1)[1 - 5(T_b/T_{pc} - 1)]\] for \[T_{pc} < T_b < 1.2T_{pc}\] and \[T_b < T_w\]

\[\bar{C}_p = \frac{H_w - H_b}{T_w - T_b}\]

Parameters

- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr** [float] Prandtl number with bulk fluid properties, [-]
- **rho_w** [float, optional] Density at the wall temperature, [kg/m^3]
- **rho_b** [float, optional] Density at the bulk temperature, [kg/m^3]
- **Cp_avg** [float, optional] Average heat capacity between the wall and bulk temperatures, [J/kg/K]
- **Cp_b** [float, optional] Heat capacity at the bulk temperature, [J/kg/K]
T_b  [float] Bulk temperature, [K]
T_w  [float] Wall temperature, [K]
T_pc [float] Pseudocritical temperature, i.e. temperature at P where Cp is at a maximum, [K]

Returns

Nu  [float] Nusselt number with bulk fluid properties, [-]

Notes

The range of examined parameters is as follows: P from 23.4 to 29.3 MPa; G from 700-3600 kg/m^2/s; q from 46 to 2600 kW/m^2; Re from 8E4 to 5E5; D from 1.6 to 20 mm.

For enhanced heat transfer database in [2], this correlation was the second best with a MAD of 11.5%. In the database in [3], the correlation was the second best as well.

This is sometimes called the Jackson-Hall correlation. If the extra information is not provided, the correlation will be used without the corrections.

References

[1], [2], [3], [4]

Examples

```python
g = Nu_Jackson(1E5, 1.2)
252.37231572974918
```

ht.conv_supercritical.Nu_Gupta(Re, Pr, rho_w=None, rho_b=None, mu_w=None, mu_b=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
Nu_w = 0.004Re^{0.923}Pr^{0.773} \left( \frac{\rho_w}{\rho_b} \right)^{0.186} \left( \frac{\mu_w}{\mu_b} \right)^{0.366}
\]

\[
\bar{C}p = \frac{H_w - H_b}{T_w - T_b}
\]

Parameters

- **Re**  [float] Reynolds number with wall fluid properties, [-]
- **Pr**  [float] Prandtl number with wall fluid properties and an average heat capacity between the wall and bulk temperatures [-]
- **rho_w** [float, optional] Density at the wall temperature, [kg/m^3]
- **rho_b** [float, optional] Density at the bulk temperature, [kg/m^3]
- **mu_w** [float, optional] Viscosity at the wall temperature, [Pa*s]
- **mu_b** [float, optional] Viscosity at the bulk temperature, [Pa*s]

Returns

- **Nu**  [float] Nusselt number with wall fluid properties, [-]
Notes

For the data used to develop the correlation, P was set at 24 MPa, and D was 10 mm. G varied from 200-1500 kg/m^2/s and q varied from 0 to 1250 kW/m^2.

Cp used in the calculation of Prandtl number should be the average value of those at the wall and the bulk temperatures.

For deteriorated heat transfer, this was the most accurate correlation in [2] with a MAD of 18.1%

If the extra density and viscosity information is not provided, it will not be used.

References

[1]. [2]

Examples

```python
>>> Nu_Gupta(1E5, 1.2, 330, 290, 8e-4, 9e-4)
186.20135477175126
```

ht.conv_supercritical.**Nu_Swenson**(Re, Pr, rho_w=None, rho_b=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
Nu_w = 0.00459 \, Re_w^{0.923} \, Pr_w^{0.613} \left( \frac{\rho_w}{\rho_b} \right)^{0.231}
\]

\[
\bar{C}_p = \frac{H_w - H_b}{T_w - T_b}
\]

Parameters

- **Re** [float] Reynolds number with wall fluid properties, [-]
- **Pr** [float] Prandtl number with wall fluid properties and an average heat capacity between the wall and bulk temperatures [-]
- **rho_w** [float, optional] Density at the wall temperature, [kg/m^3]
- **rho_b** [float, optional] Density at the bulk temperature, [kg/m^3]

Returns

- **Nu** [float] Nusselt number with wall fluid properties, [-]

Notes

The range of examined parameters is as follows: P from 22.8 to 27.6 MPa; G from 542-2150 kg/m^2/s; Re from 7.5E4 to 3.16E6; T_b from 75 to 576 degrees Celsius and T_w from 93 to 649 degrees Celsius.

Cp used in the calculation of Prandtl number should be the average value of those at the wall and the bulk temperatures.

For deteriorated heat transfer, this was the most accurate correlation in [2] with a MAD of 18.4%. On the overall database in [3], it was the 9th most accurate correlation.

If the extra density information is not provided, it will not be used.
References

[1], [2], [3], [4]

Examples

```python
>>> Nu_Swenson(1E5, 1.2, 330, 290.)
217.92827034803668
```

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
Nu_b = 0.02269 Re_b^{0.8079} Pr_b^{0.9213} \left( \frac{\rho_w}{\rho_b} \right)^{0.6638} \left( \frac{\mu_w}{\mu_b} \right)^{0.8687}
\]

\[
\bar{C}_p = \frac{H_w - H_b}{T_w - T_b}
\]

Parameters

- `Re` [float] Reynolds number with bulk fluid properties, [-]
- `Pr` [float] Prandtl number with bulk fluid properties and an average heat capacity between the wall and bulk temperatures [-]
- `rho_w` [float, optional] Density at the wall temperature, [kg/m^3]
- `rho_b` [float, optional] Density at the bulk temperature, [kg/m^3]
- `mu_w` [float, optional] Viscosity at the wall temperature, [Pa*s]
- `mu_b` [float, optional] Viscosity at the bulk temperature, [Pa*s]

Returns

- `Nu` [float] Nusselt number with bulk fluid properties, [-]

Notes

For the data used to develop the correlation, P varied from 23 to 30 MPa, and D was 12 mm. G varied from 600-1200 kg/m^2/s and q varied from 100 to 600 kW/m^2.

Cp used in the calculation of Prandtl number should be the average value of those at the wall and the bulk temperatures.

For deteriorated heat transfer, this was the third most accurate correlation in [2] with a MAD of 20.5%.

If the extra density and viscosity information is not provided, it will not be used.

References

[1], [2]

Examples
ht.conv_supercritical.Nu_Mokry(Re, Pr, rho_w=None, rho_b=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1], and reviewed in [2].

\[
N_{ub} = 0.0061 Re_b^{0.904} Pr_b^{0.684} \left( \frac{\rho_w}{\rho_b} \right)^{0.564}
\]

Parameters

Re [float] Reynolds number with bulk fluid properties, [-]
Pr [float] Prandtl number with bulk fluid properties and an average heat capacity between the wall and bulk temperatures [-]
rho_w [float, optional] Density at the wall temperature, [kg/m^3]
rho_b [float, optional] Density at the bulk temperature, [kg/m^3]

Returns

Nu [float] Nusselt number with bulk fluid properties, [-]

Notes

For the data used to develop the correlation, P was set at 20 MPa, and D was 10 mm. G varied from 200-1500 kg/m^2/s and q varied from 0 to 1250 kW/m^2.

Cp used in the calculation of Prandtl number should be the average value of those at the wall and the bulk temperatures.

For deteriorated heat transfer, this was the four most accurate correlation in [2] with a MAD of 24.0%. It was also the 7th most accurate against enhanced heat transfer, with a MAD of 14.7%, and the most accurate for the normal heat transfer database as well as the top correlation in all categories combined.

If the extra density information is not provided, it will not be used.

References

[1], [2]

Examples

```python
>>> Nu_Mokry(1E5, 1.2, 330, 290.)
246.11563191569923
```

ht.conv_supercritical.Nu_Bringer_Smith(Re, Pr)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under near-supercritical conditions according to [1] and as shown in [2] and [3].

\[
N_{ux} = 0.0266 Re_x^{0.77} Pr_w^{0.55}
\]
Parameters

Re [float] Reynolds number with fluid properties at T_ref, [-]
Pr [float] Prandtl number with wall fluid properties, [-]

Returns

Nu [float] Nusselt number with fluid properties at T_ref, [-]

Notes

Fit to data somewhat distant from the critical and pseudo-critical regions. Found to fit the data in [3] fourth best; in [2] however, it was ranked so low that no ranking was given.

T_ref and the properties therein should be evaluated as follows:

\[
T_{ref} = T_b \text{ if } \frac{T_{pc} - T_b}{T_w - T_b} < 0
\]

\[
T_{ref} = T_{pc} \text{ if } 0 < \frac{T_{pc} - T_b}{T_w - T_b} < 1
\]

\[
T_{ref} = T_w \text{ if } \frac{T_{pc} - T_b}{T_w - T_b} > 1
\]

References

[1], [2], [3]

Examples

```python
>>> Nu_Bringer_Smith(1E5, 1.2)
208.17631753279107
```

ht.conv_supercritical.Nu_Ornatsky(Re, Pr_b, Pr_w, rho_w=None, rho_b=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1] as shown in both [2] and [3].

\[
Nu_b = 0.023Re_b^{0.8}(\min(Pr_b, Pr_w))^{0.8}(\frac{\rho_w}{\rho_b})^{0.3}
\]

Parameters

Re [float] Reynolds number with bulk fluid properties, [-]
Pr_b [float] Prandtl number with bulk fluid properties, [-]
Pr_w [float] Prandtl number with wall fluid properties, [-]
rho_w [float, optional] Density at the wall temperature, [kg/m^3]
rho_b [float, optional] Density at the bulk temperature, [kg/m^3]

Returns

Nu [float] Nusselt number with bulk fluid properties, [-]
Notes

[2] ranked it thirteenth in the enhanced heat transfer category, with a MAD of 19.8% and 11th in the normal heat transfer with a MAD of 17.6%. [3] ranked it seventh on a combined database.

If the extra density information is not provided, it will not be used.

References

[1], [2], [3]

Examples

```python
g = Nu_Ornatsky(1E5, 1.2, 1.5, 330, 290.)
276.63531150832307
g = Nu_Gorban(Re, Pr)
ht.conv_supercritical.Nu_Gorban(Re, Pr)
Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1]. Not recommended.

\[ Nu_b = 0.0059 Re_b^{0.90} Pr_b^{-0.12} \]

Parameters

- Re [float] Reynolds number with bulk fluid properties, [-]
- Pr [float] Prandtl number with bulk fluid properties, [-]

Returns

- Nu [float] Nusselt number with bulk fluid properties, [-]

Notes


References

[1], [2], [3]

Examples

```python
g = Nu_Gorban(1E5, 1.2)
182.5367282733999
```

ht.conv_supercritical.Nu_Zhu(Re, Pr, rho_w=None, rho_b=None, k_w=None, k_b=None)
Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[ Nu_b = 0.0068 Re_b^{0.9} Pr_b^{0.63} \left( \frac{\rho_w}{\rho_b} \right)^{0.17} \left( \frac{k_w}{k_b} \right)^{0.29} \]

\[ \tilde{C}_p = \frac{H_w - H_b}{T_w - T_b} \]

2.14. Convection with supercritical fluids (ht.conv_supercritical)
Parameters

- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr** [float] Prandtl number with bulk fluid properties and an average heat capacity between the wall and bulk temperatures [-]
- **rho_w** [float, optional] Density at the wall temperature, [kg/m^3]
- **rho_b** [float, optional] Density at the bulk temperature, [kg/m^3]
- **k_w** [float, optional] Thermal conductivity at the wall temperature, [W/m/K]
- **k_b** [float, optional] Thermal conductivity at the bulk temperature, [W/m/K]

Returns

- **Nu** [float] Nusselt number with bulk fluid properties, [-]

Notes

For the data used to develop the correlation, P varied from 22 to 30 MPa, and D was 26 mm. G varied from 600-1200 kg/m^2/s and q varied from 200 to 600 kW/m^2.

$C_p$ used in the calculation of Prandtl number should be the average value of those at the wall and the bulk temperatures.

On the overall database in [2], this was the 8th most accurate correlation, and ninth most accurate against normal heat transfer.

If the extra density and thermal conductivity information is not provided, it will not be used.

References

[1], [2]

Examples

```python
>>> Nu_Zhu(1E5, 1.2, 330, 290., 0.63, 0.69)
240.1459854494706
```

ht.conv_supercritical.Nu_Bishop(Re, Pr, rho_w=None, rho_b=None, D=None, x=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1]. Correlation includes an adjustment for the thermal entry length. One of the most common correlations for supercritical convection.

$$Nu_b = 0.0069Re_b^{0.9} Pr_b^{0.66} \left( \frac{\rho_w}{\rho_b} \right)^{0.43} (1 + 2.4D/x)$$

$$\bar{C}_p = \frac{H_w - H_b}{T_w - T_b}$$

Parameters

- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr** [float] Prandtl number with bulk fluid properties and an average heat capacity between the wall and bulk temperatures [-]
- **rho_w** [float, optional] Density at the wall temperature, [kg/m^3]
Heat Transfer Documentation, Release 0.1.56

**rho_b** [float, optional] Density at the bulk temperature, [kg/m³]

**D** [float, optional] Diameter of tube, [m]

**x** [float, optional] Axial distance along the tube, [m]

**Returns**

**Nu** [float] Nusselt number with wall fluid properties, [-]

**Notes**

For the data used to develop the correlation, P varied from 22.8 to 27.6 MPa, and D was x/D varied from 30–365. G varied from 651–3662 kg/m²/s and q varied from 310 to 3460 kW/m². T_b varied from 282 to 527 degrees Celsius.

Cp used in the calculation of Prandtl number should be the average value of those at the wall and the bulk temperatures.

For enhanced heat transfer, this was the 11th most accurate correlation in [2] with a MAD of 19.0%. On the overall database in [3], it was the most accurate correlation however.

If the extra density information is not provided, it will not be used. If both diameter and axial distance are not provided, the entrance correction is not used.

**References**

[1], [2], [3], [4]

**Examples**

```python
>>> Nu_Bishop(1E5, 1.2, 330, 290., .01, 1.2)
265.3620050072533
```

ht.conv_supercritical.Nu_Yamagata (Re, Pr, Pr_pc=None, Cp_avg=None, Cp_b=None, T_b=None, T_w=None, T_pc=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
N_u_b = 0.0138 \bar{Re}_b^{0.85} Pr_b^{0.8} F
\]

\[
F = \left( \frac{\bar{C}_p}{\bar{C}_{p,b}} \right)^{n_2} \text{ if } \frac{T_{pc} - T_b}{T_w - T_b} < 0
\]

\[
F = 0.67 \bar{Pr}_{pc}^{-0.05} \left( \frac{\bar{C}_p}{\bar{C}_{p,b}} \right)^{n_1} \text{ if } 0 < \frac{T_{pc} - T_b}{T_w - T_b} < 1
\]

\[
F = 1 \text{ if } \frac{T_{pc} - T_b}{T_w - T_b} > 1
\]

\[
n_1 = -0.77(1 + 1/Pr_{pc}) + 1.49
\]

\[
n_2 = 1.44(1 + 1/Pr_{pc}) - 0.53
\]

\[
\bar{C}_p = \frac{H_w - H_b}{T_w - T_b}
\]

**Parameters**

2.14. Convection with supercritical fluids (ht.conv_supercritical) 139
Re [float] Reynolds number with bulk fluid properties, [-]
Pr [float] Prandtl number with bulk fluid properties, [-]
Pr_pc [float, optional] Prandtl number at the pseudocritical temperature, [-]
Cp_avg [float, optional] Average heat capacity between the wall and bulk temperatures, [J/kg/K]
Cp_b [float, optional] Heat capacity at the bulk temperature, [J/kg/K]
T_b [float] Bulk temperature, [K]
T_w [float] Wall temperature, [K]
T_pc [float] Pseudocritical temperature, i.e. temperature at P where Cp is at a maximum, [K]

Returns
Nu [float] Nusselt number with bulk fluid properties, [-]

Notes
For the data used to develop the correlation, P varied from 22.6 to 29.4 MPa, and D was 7.5 and 10 mm. G varied from 310-1830 kg/m^2/s, q varied from 116 to 930 kW/m^2, and bulk temperature varied from 230 to 540 degrees Celsius.
In the database in [3], the correlation was considered but not tested. In [2], the correlation was considered but no results were reported.
For enhanced heat transfer database in [2], this correlation was the second best with a MAD of 11.5%. In the database in [3], the correlation was the second best as well.
If the extra information is not provided, the correlation will be used without the corrections.

References
[1], [2], [3], [4]

Examples

```python
>>> Nu_Yamagata(Re=1E5, Pr=1.2, Pr_pc=1.5, Cp_avg=2080.845, Cp_b=2048.621, T_b=650, T_w=700, T_pc=600.0)
292.3473428004679
```

ht.conv_supercritical.Nu_Kitoh(Re, Pr, H=None, G=None, q=None)
Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1], also shown in [2], [3] and [4]. Depends on fluid enthalpy, mass flux, and heat flux.

\[
N_u = 0.015Re_0^{0.85}Pr_b^m \\
m = 0.69 - \frac{81000}{q_{dht}} + f_c q \\
q_{dht} = 200G^{1.2} \\
f_c = 2.9 \times 10^{-8} + \frac{0.11}{q_{dht}} \text{ for } H_b < 1500 \text{ kJ/kg}
\]
\[ f_c = -8.7 \times 10^{-8} - \frac{0.65}{q_{db}} \text{ for } 1500 \text{ kJ/kg} < H_b < 3300 \text{ kJ/kg} \]
\[ f_c = -9.7 \times 10^{-7} + \frac{1.3}{q_{db}} \text{ for } H_b > 3300 \text{ kJ/kg} \]

Parameters

- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr** [float] Prandtl number with bulk fluid properties, [-]
- **H** [float, optional] Enthalpy of water (if the fluid is water), [J/kg]
- **G** [float, optional] Mass flux of the fluid, [kg/m^2/s]
- **q** [float, optional] Heat flux to wall, [W/m^2]

Returns

- **Nu** [float] Nusselt number as explained below, [-]

Notes

The reference point for the enthalpy values is not stated in [1]. The upper and lower enthalpy limits for this correlation are 4000 kJ/kg and 0 kJ/kg, but these are not enforced in this function.

If not all of H, G, and q are provided, the correlation is used without the correction.

This correlation was ranked 6th best in [3], and found 4th best for enhanced heat transfer in [2] with a MAD of 12.3%.

For the data used to develop the correlation, G varied from 100-1750 kg/m^2/s, q varied from 0 to 1800 kW/m^2, and bulk temperature varied from 20 to 550 degrees Celsius.

This correlation does not have realistic behavior for values outside those used in the study, and should not be used.

References

[1], [2], [3], [4]

Examples

```python
>>> Nu_Kitoh(1E5, 1.2, 1.3E6, 1500, 5E6)
331.80234139591306
```

ht.conv_supercritical.Nu_Krasnoshchekov_Protopopov (Re, Pr, Cp_avg=None, Cp_b=None, k_w=None, k_b=None, mu_w=None, mu_b=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
Nu_b = Nu_0 \left( \frac{\mu_w}{\mu_b} \right)^{0.11} \left( \frac{k_b}{k_w} \right)^{-0.33} \left( \frac{C_p}{C_{p,b}} \right)^{0.35}
\]

\[
Nu_0 = \frac{(f/8)Re_b Pr_b}{1.07 + 12.7(f/8)^{1/2}(Pr_b)^{2/3} - 1}
\]

\[
f d = [1.82 \log_{10}(Re_b) - 1.64]^2
\]
Parameters

- Re [float] Reynolds number with bulk fluid properties, [-]
- Pr [float] Prandtl number with bulk fluid properties [-]
- Cp_avg [float, optional] Average heat capacity between the wall and bulk temperatures, [J/kg/K]
- Cp_b [float, optional] Heat capacity at the bulk temperature, [J/kg/K]
- k_w [float, optional] Thermal conductivity at the wall temperature, [W/m/K]
- k_b [float, optional] Thermal conductivity at the bulk temperature, [W/m/K]
- mu_w [float, optional] Viscosity at the wall temperature, [Pa*s]
- mu_b [float, optional] Viscosity at the bulk temperature, [Pa*s]

Returns

- Nu [float] Nusselt number with bulk fluid properties, [-]

Notes

For the data used to develop the correlation, P varied from 22.3 to 32 MPa, Re varied from 2E4 to 8.6E6, Pr from 0.86-86, viscosity ratio from 0.9 to 3.6, thermal conductivity ratio from 1 to 6, and heat capacity ratio from 0.07 to 4.5.

For the heat transfer database in [3], this correlation was 14th most accurate.

If the extra heat capacity, viscosity, and thermal conductivity information is not provided, it will not be used.

References

[1], [2], [3], [4]

Examples

```python
>>> Nu_Krasnoshchekov_Protopopov(1E5, 1.2, 330, 290., 0.62, 0.52, 8e-4, 9e-4)
228.85296737400222
```

ht.conv_supercritical.Nu_Petukhov(Re, Pr, rho_w=None, rho_b=None, mu_w=None, mu_b=None)

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
Nu_b = \frac{(f/8)Re_bPr_b}{1 + 900/Re_b + 12.7(f/8)^{1/2}(Pr_b)^{2/3} - 1}
\]

\[
f = f_d \left( \frac{\rho_w}{\rho_b} \right)^{0.4} \left( \frac{\mu_w}{\mu_b} \right)^{0.2}
\]

\[
f_d = [1.82 \log_{10}(Re_b) - 1.64]^{-2}
\]

Parameters

- Re [float] Reynolds number with bulk fluid properties, [-]
- Pr [float] Prandtl number with bulk fluid properties [-]
**rho_w** [float, optional] Density at the wall temperature, [kg/m³]

**rho_b** [float, optional] Density at the bulk temperature, [kg/m³]

**mu_w** [float, optional] Viscosity at the wall temperature, [Pa·s]

**mu_b** [float, optional] Viscosity at the bulk temperature, [Pa·s]

**Returns**

**Nu** [float] Nusselt number with bulk fluid properties, [-]

**Notes**

For the heat transfer database in [2], this correlation was 5th most accurate in the enhanced heat transfer category, and second in the normal heat transfer category with MADs of 13.8% and 12.0% respectively.

If the extra viscosity and density information is not provided, it will not be used.

**References**

[1], [2]

**Examples**

```python
>>> Nu_Petukhov(1E5, 1.2, 330, 290., 8e-4, 9e-4)
254.8258598466738
```

**ht.conv_supercritical.Nu_Krasnoshchekov**

```
Re, Pr, rho_w=None, rho_b=None,
Cp_avg=None, Cp_b=None, T_b=None,
T_w=None, T_pc=None)
```

Calculates internal convection Nusselt number for turbulent vertical upward flow in a pipe under supercritical conditions according to [1].

\[
N_u_b = N_u_0 \left( \frac{\rho_w}{\rho_b} \right)^{0.3} \left( \frac{C_p}{C_p,b} \right)^n
\]

\[
N_u_0 = \frac{(f/8)Re_bPr_b}{1.07 + 12.7(f/8)^{1/2}(Pr_b^{2/3} - 1)}
\]

\[
f_d = [1.82 \log_{10}(Re_b) - 1.64]^{-2}
\]

\[n = 0.4 \text{ for } T_b < T_w < T_{pc} \text{ or } 1.2 T_{pc} < T_b < T_w\]

\[n = n_1 = 0.22 + 0.18 T_w/T_{pc} \text{ for } 1 < T_w/T_{pc} < 2.5\]

\[n = n_1 + (5n_1 - 2)(1 - T_b/T_{pc}) \text{ for } T_{pc} < T_b < 1.2 T_{pc} \text{ and } T_b < T_w\]

**Parameters**

- **Re** [float] Reynolds number with bulk fluid properties, [-]
- **Pr** [float] Prandtl number with bulk fluid properties, [-]
- **rho_w** [float, optional] Density at the wall temperature, [kg/m³]
- **rho_b** [float, optional] Density at the bulk temperature, [kg/m³]
\[ \Delta P = \frac{f(m/S_s)^2 D_s (N_B + 1)}{2 \rho D_e (\mu/\mu_w)^{0.14}} \]

\[ S_s = \frac{D_s (P_T - D_o) L_B}{P_T} \]

\[ D_e = \frac{4(P_T^2 - \pi D_o^2/4)}{\pi D_o} \]

**Parameters**

- **m** [float] Mass flow rate, [kg/s]
- **rho** [float] Fluid density, [kg/m^3]
- **mu** [float] Fluid viscosity, [Pa*s]
- **DShell** [float] Diameter of exchanger shell, [m]
**LSpacing** [float] Baffle spacing, [m]

**pitch** [float] Tube pitch, [m]

**Do** [float] Tube outer diameter, [m]

**NBaffles** [float] Baffle count, []

**mu_w** [float] Fluid viscosity at wall temperature, [Pa*s]

**Returns**

- **dP** [float] Pressure drop across bundle, [Pa]

**Notes**

Adjustment for viscosity left out of this function. Example is from [2]. Roughly 10% difference due to reading of graph. Graph scanned from [1], and interpolation is used to read it.

**References**

[1], [2]

**Examples**

```python
>>> dP_Kern(m=11., rho=995., mu=0.000803, mu_w=0.000657, DShell=0.584,
... LSpacing=0.1524, pitch=0.0254, Do=.019, NBaffles=22)
18980.58768759033
```

ht.conv_tube_bank.dP_Zukauskas (**Re**, **n**, **ST**, **SL**, **D**, **rho**, **Vmax**)  
Calculates pressure drop for crossflow across a tube bank of tube number n at a specified Re. Method presented in [1]. Also presented in [2].

\[
\Delta P = N_L \chi \left( \frac{\rho V_{max}^2}{2} \right) f
\]

**Parameters**

- **Re** [float] Reynolds number, [-]
- **n** [float] Number of tube rows, [-]
- **ST** [float] Transverse pitch, used only by some conditions, [m]
- **SL** [float] Longitudal pitch, used only by some conditions, [m]
- **D** [float] Tube outer diameter, [m]
- **rho** [float] Fluid density, [kg/m^3]
- **Vmax** [float] Maximum velocity, [m/s]

**Returns**

- **dP** [float] Pressure drop, [Pa]

**Notes**

Does not account for effects in a heat exchanger. Example 2 is from [2]. Matches to 0.3%; figures are very approximate. Interpolation used with 4 graphs to obtain friction factor and a correction factor.
References

[1], [2]

Examples

```python
>>> dP_Zukauskas(Re=13943., n=7, ST=0.0313, SL=0.0343, D=0.0164, rho=1.217, Vmax=12.6)
235.22916169118335
>>> dP_Zukauskas(Re=13943., n=7, ST=0.0313, SL=0.0313, D=0.0164, rho=1.217, Vmax=12.6)
217.0750033117563
```

ht.conv_tube_bank.Nu_ESDU_73031(Re, Pr, tube_rows, pitch_parallel, pitch_normal, Pr_wall=None, angle=90.0)

Calculates the Nusselt number for crossflow across a tube bank with a specified number of tube rows, at a specified Re according to [1], also shown in [2].

\[ \text{Nu} = a\text{Re}^m\text{Pr}^{0.34}F_1F_2 \]

The constants \(a\) and \(m\) come from the following tables:

In-line tube banks:

<table>
<thead>
<tr>
<th>Re</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-300</td>
<td>0.742</td>
<td>0.431</td>
</tr>
<tr>
<td>300-2E5</td>
<td>0.211</td>
<td>0.651</td>
</tr>
<tr>
<td>2E5-2E6</td>
<td>0.116</td>
<td>0.700</td>
</tr>
</tbody>
</table>

Staggered tube banks:

<table>
<thead>
<tr>
<th>Re</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-300</td>
<td>1.309</td>
<td>0.360</td>
</tr>
<tr>
<td>300-2E5</td>
<td>0.273</td>
<td>0.635</td>
</tr>
<tr>
<td>2E5-2E6</td>
<td>0.124</td>
<td>0.700</td>
</tr>
</tbody>
</table>

Parameters

- **Re** [float] Reynolds number with respect to average (bulk) fluid properties and tube outside diameter, [-]
- **Pr** [float] Prandtl number with respect to average (bulk) fluid properties, [-]
- **tube_rows** [int] Number of tube rows per bundle, [-]
- **pitch_parallel** [float] Distance between tube center along a line parallel to the flow; has been called *longitudinal* pitch, \( pp, s2, SL, \) and \( p2 \), [m]
- **pitch_normal** [float] Distance between tube centers in a line 90° to the line of flow; has been called the *transverse* pitch, \( pn, s1, ST, \) and \( p1 \), [m]
- **Pr_wall** [float, optional] Prandtl number at the wall temperature; provide if a correction with the defaults parameters is desired; otherwise apply the correction elsewhere, [-]
- **angle** [float, optional] The angle of inclination of the tube bank with respect to the longitudinal axis (90° for a straight tube bank)
Returns

\( \text{Nu} \) [float] Nusselt number with respect to tube outside diameter, [-]

See also:

ESDU\_tube\_angle\_correction
ESDU\_tube\_row\_correction

Notes

The tube-row count correction factor \( F2 \) can be disabled by setting \( \text{tube\_rows} \) to 10. The property correction factor \( F1 \) can be disabled by not specifying \( Pr\_wall \). A Prandtl number exponent of 0.26 is recommended in [1] for heating and cooling for both liquids and gases.

The pitches are used to determine whether or not to use data for staggered or inline tube banks.

The inline coefficients are valid for a normal pitch to tube diameter ratio from 1.2 to 4; and the staggered ones from 1 to 4. The overall accuracy of this method is claimed to be 15%.

References

[1], [2]

Examples

```python
>>> Nu_ESDU_73031(Re=1.32E4, Pr=0.71, tube_rows=8, pitch_parallel=.09,
... pitch_normal=.05)
98.2563319140594
```

ht.conv\_tube\_bank.Nu\_Zukauskas\_Bejan(Re, Pr, tube\_rows, pitch\_parallel, pitch\_normal, Pr\_wall=None)

Calculates Nusselt number for crossflow across a tube bank of tube number \( n \) at a specified \( Re \) according to the method of Zukauskas [1]. A fit to graphs from [1] published in [2] is used for the correlation. The tube row correction factor is obtained from digitized graphs from [1], and a lookup table was created and is used for speed.

The formulas are as follows:

Aligned tube banks:

\[
\bar{\text{Nu}}_D = 0.9C_n R_e^{0.4} P_r^{0.36} \left( \frac{P_r}{P_{r_w}} \right)^{0.25} \text{ for } 1 < R_e < 100
\]

\[
\bar{\text{Nu}}_D = 0.52C_n R_e^{0.5} P_r^{0.36} \left( \frac{P_r}{P_{r_w}} \right)^{0.25} \text{ for } 100 < R_e < 1000
\]

\[
\bar{\text{Nu}}_D = 0.27C_n R_e^{0.63} P_r^{0.36} \left( \frac{P_r}{P_{r_w}} \right)^{0.25} \text{ for } 1000 < R_e < 20000
\]

\[
\bar{\text{Nu}}_D = 0.033C_n R_e^{0.8} P_r^{0.36} \left( \frac{P_r}{P_{r_w}} \right)^{0.25} \text{ for } 20000 < R_e < 200000
\]

2.15. Heat transfer and pressure drop across tube bundles (ht.conv\_tube\_bank)
Staggered tube banks:

\[
\tilde{N}u_D = 1.04C_nRe_D^{0.4}Pr^{0.36}\left(\frac{Pr}{Pr_w}\right)^{0.25} \quad \text{for } 1 < Re < 500
\]

\[
\tilde{N}u_D = 0.71C_nRe_D^{0.5}Pr^{0.36}\left(\frac{Pr}{Pr_w}\right)^{0.25} \quad \text{for } 500 < Re < 1000
\]

\[
\tilde{N}u_D = 0.35C_nRe_D^{0.6}Pr^{0.36}\left(\frac{Pr}{Pr_w}\right)^{0.25}\left(\frac{X_t}{X_l}\right)^{0.2} \quad \text{for } 1000 < Re < 20000
\]

\[
\tilde{N}u_D = 0.031C_nRe_D^{0.8}Pr^{0.36}\left(\frac{Pr}{Pr_w}\right)^{0.25}\left(\frac{X_t}{X_l}\right)^{0.2} \quad \text{for } 20000 < Re < 200000
\]

Parameters

- **Re** [float] Reynolds number with respect to average (bulk) fluid properties and tube outside diameter, [-]
- **Pr** [float] Prandtl number with respect to average (bulk) fluid properties, [-]
- **tube_rows** [int] Number of tube rows per bundle, [-]
- **pitch_parallel** [float] Distance between tube center along a line parallel to the flow; has been called longitudinal pitch, pp, s2, SL, and p2, [m]
- **pitch_normal** [float] Distance between tube centers in a line 90° to the line of flow; has been called the transverse pitch, pn, s1, ST, and p1, [m]
- **Pr_wall** [float, optional] Prandtl number at the wall temperature; provide if a correction with the defaults parameters is desired; otherwise apply the correction elsewhere, [-]

Returns

- **Nu** [float] Nusselt number with respect to tube outside diameter, [-]

Notes

If **Pr_wall** is not provided, the Prandtl number correction is not used and left to an outside function. A Prandtl number exponent of 0.25 is recommended in [1] for heating and cooling for both liquids and gases.

References

[1], [2]

Examples

```python
>>> Nu_Zukauskas_Bejan(Re=1E4, Pr=7., tube_rows=10, pitch_parallel=.05, pitch_normal=.05)
175.9202277145248
```

ht.conv_tube_bank.Nu_HEDH_tube_bank(Re, Pr, Do, tube_rows, pitch_parallel, pitch_normal)

Calculates Nusselt number for crossflow across a tube bank of tube rows at a specified **Re**, **Pr**, and **D** using the Heat Exchanger Design Handbook (HEDH) methodology, presented in [1].

\[
Nu = Nu_m f_N
\]
\[ Nu_m = 0.3 + \sqrt{Nu_{m,\text{lam}}^2 + Nu_{m,\text{turb}}^2} \]

\[
Nu_{m,\text{turb}} = \frac{0.037Re^{0.8}Pr}{1 + 2.443Re^{-0.1}(Pr^{2/3} - 1)} 
\]

\[ Nu_{m,\text{lam}} = 0.664Re^{0.5}Pr^{1/3} \]

\[
\psi = 1 - \frac{\pi}{4a} \text{ if } b \geq 1 \\
\psi = 1 - \frac{\pi}{4ab} \text{ if } b < 1 
\]

\[ f_A = 1 + \frac{0.7}{\psi^{1.5}} \left( \frac{b}{a} + 0.7 \right)^2 \text{ if inline} \]

\[ f_A = 1 + \frac{2}{3b} \text{ elif partly staggered} \]

\[ f_N = \frac{1 + (n - 1)f_A}{n} \]

**Parameters**

- Re [float] Reynolds number with respect to average (bulk) fluid properties and tube outside diameter, [-]
- Pr [float] Prandtl number with respect to average (bulk) fluid properties, [-]
- Do [float] Tube outer diameter, [m]
- tube_rows [int] Number of tube rows per bundle, [-]
- pitch_parallel [float] Distance between tube center along a line parallel to the flow; has been called longitudinal pitch, pp, s2, SL, and p2, [m]
- pitch_normal [float] Distance between tube centers in a line 90° to the line of flow; has been called the transverse pitch, pn, s1, ST, and p1, [m]

**Returns**

- Nu [float] Nusselt number with respect to tube outside diameter, [-]

**Notes**

Prandtl number correction left to an outside function, although a set of coefficients were specified in [1] because they depend on whether heating or cooling is happening, and for gases, use a temperature ratio instead of Prandtl number.

The claimed range of validity of these expressions is \(10 < Re < 1E5\) and \(0.6 < Pr < 1000\).

**References**

[1], [2]
Examples

```python
>>> Nu_HEDH_tube_bank(Re=1E4, Pr=7., tube_rows=10, pitch_normal=.05,
... pitch_parallel=.05, Do=.03)
382.4636554404698
```

Example 3.11 in [2]:

```python
>>> Nu_HEDH_tube_bank(Re=10263.37, Pr=.708, tube_rows=11, pitch_normal=.05,
... pitch_parallel=.05, Do=.025)
149.18735251017594
```

ht.conv_tube_bank.`Nu_Grimison_tube_bank` *(Re, Pr, Do, tube_rows, pitch_parallel,
pitch_normal)*

Calculates Nusselt number for crossflow across a tube bank of tube rows at a specified $Re$, $Pr$, and $D$ using the Grimison methodology as described in [1].

\[
\overline{Nu}_D = 1.13C_1Re_{D,\text{max}}^{m}Pr^{1/3}C_2
\]

Parameters

- **Re** [float] Reynolds number with respect to average (bulk) fluid properties and tube outside diameter, [-]
- **Pr** [float] Prandtl number with respect to average (bulk) fluid properties, [-]
- **Do** [float] Tube outer diameter, [m]
- **tube_rows** [int] Number of tube rows per bundle, [-]
- **pitch_parallel** [float] Distance between tube center along a line parallel to the flow; has been called *longitudinal* pitch, $pp$, $s2$, $SL$, and $p2$, [m]
- **pitch_normal** [float] Distance between tube centers in a line $90^\circ$ to the line of flow; has been called the *transverse* pitch, $pn$, $s1$, $ST$, and $p1$, [m]

Returns

- **Nu** [float] Nusselt number with respect to tube outside diameter, [-]

Notes

Tube row correction factors are applied for tube row counts less than 10, also published in [1].

References

[1]

Examples

```python
>>> Nu_Grimison_tube_bank(Re=10263.37, Pr=.708, tube_rows=11,
... pitch_normal=.05, pitch_parallel=.05, Do=.025)
79.07883866010096
```

```python
>>> Nu_Grimison_tube_bank(Re=10263.37, Pr=.708, tube_rows=11,
... pitch_normal=.07, pitch_parallel=.05, Do=.025)
79.92721078571385
```
ht.conv_tube_bank.Zukauskas_tube_row_correction(tube_rows, staggered=True, 
Re=10000.0)

Calculates the tube row correction factor according to a graph digitized from [1] and also shown in [2] for heat transfer across a tube bundle. The correction factors are slightly different for staggered vs. inline configurations; for the staggered configuration, factors are available separately for Re larger or smaller than 1000.

This method is a tabular lookup, with values of 1 when the tube row count is 20 or more.

Parameters

- **tube_rows** [int] Number of tube rows per bundle, [-]
- **staggered** [bool, optional] Whether in the in-line or staggered configuration, [-]
- **Re** [float, optional] The Reynolds number of flow through the tube bank using the bare tube outer diameter and the minimum flow area through the bundle, [-]

Returns

- **F** [float] Tube row count correction factor, [-]

Notes

The basis for this method is that an infinitely long tube bank has a factor of 1; in practice the factor is reached at 20 rows.

References

[1]

Examples

```python
>>> Zukauskas_tube_row_correction(4, staggered=True)
0.8942
>>> Zukauskas_tube_row_correction(6, staggered=False)
0.9465
```

ht.conv_tube_bank.ESDU_tube_row_correction(tube_rows, staggered=True, 
Re=3000.0, 
method='Hewitt')

Calculates the tube row correction factor according to [1] as shown in [2] for heat transfer across a tube bundle. This is also used for finned bundles. The correction factors are slightly different for staggered vs. inline configurations.

This method is a tabular lookup, with values of 1 when the tube row count is 10 or more.

Parameters

- **tube_rows** [int] Number of tube rows per bundle, [-]
- **staggered** [bool, optional] Whether in the in-line or staggered configuration, [-]
- **Re** [float, optional] The Reynolds number of flow through the tube bank using the bare tube outer diameter and the minimum flow area through the bundle, [-]
- **method** [str, optional] ‘Hewitt’; this may have another option in the future, [-]

Returns

- **F2** [float] ESDU tube row count correction factor, [-]
Notes

In [1], for line data, there are two curves given for different Reynolds number ranges. This is not included in [2] and only an average curve is given. This is not implemented here; $Re$ is an argument but does not impact the result of this function.

For tube counts 1-7, [3] claims the factors from [1] are on average: $[0.65, 0.77, 0.84, 0.9, 0.94, 0.97, 0.99]$.

References

[1], [2], [3]

Examples

```python
>>> ESDU_tube_row_correction(4, staggered=True)
0.8984
>>> ESDU_tube_row_correction(6, staggered=False)
0.9551
```

ht.conv_tube_bank.ESDU_tube_angle_correction(angle)
Calculates the tube bank inclination correction factor according to [1] for heat transfer across a tube bundle.

$$F_3 = \frac{Nu_\theta}{Nu_{\theta=90}} = (\sin(\theta))^{0.6}$$

Parameters

- angle [float] The angle of inclination of the tube bank with respect to the longitudinal axis (90° for a straight tube bank)

Returns

- $F_3$ [float] ESDU tube inclination correction factor, [-]

Notes

A curve is given in [1] but it is so close the function, it is likely the function is all that is used. [1] claims this correction is valid for $100 < Re < 10^6$.

For angles less than 10°, the problem should be considered internal flow, not flow across a tube bank.

References

[1]

Examples

```python
>>> ESDU_tube_angle_correction(75)
0.9794139080247666
```

ht.conv_tube_bank.baffle_correction_Bell(crossflow_tube_fraction, method='spline')
Calculate the baffle correction factor $J_c$ which accounts for the fact that all tubes are not in crossflow to the fluid - some have fluid flowing parallel to them because they are situated in the “window”, where the baffle is cut, instead of between the tips of adjacent baffles.
Equal to 1 for no tubes in the window, increases to 1.15 when the windows are small and velocity there is high; decreases to about 0.52 for very large baffle cuts. Well designed exchangers should typically have a value near 1.0.

Cubic spline interpolation is the default method of retrieving a value from the graph, which was digitized with Engauge-Digitizer.

The interpolation can be slightly slow, so a Chebyshev polynomial was fit to a maximum error of 0.142%, average error 0.04% - well within the margin of error of the digitization of the graph; this is approximately 10 times faster, accessible via the ‘chebyshev’ method.

The Heat Exchanger Design Handbook [4], [5] provides the linear curve fit, which covers the “practical” range of baffle cuts 15-45% but not the last dip in the graph. This method is not recommended, but can be used via the method “HEDH”.

\[ J_c = 0.55 + 0.72 F_c \]

\textbf{crossflow Tube Fraction} [float] Fraction of tubes which are between baffle tips and not in the window, [-]

\textbf{method} [str, optional] One of ‘chebyshev’, ‘spline’, or ‘HEDH’

\textbf{Returns}

\[ J_c \] [float] Baffle correction factor in the Bell-Delaware method, [-]

\textbf{Notes}

max: ~1.1536 at ~0.9066 min: ~0.5328 at 0 value at 1: ~1.0314

For the ‘spline’ method, this function takes ~13 us per call. The other two methods are approximately 10x faster.

\textbf{References}

[1], [2], [3], [4], [5]

\textbf{Examples}

For a HX with four groups of tube bundles; the top and bottom being 9 tubes each, in the window, and the two middle bundles having 41 tubes each, for a total of 100 tubes, the fraction between baffle tubes and not in the window is 0.82. The correction factor is then:

```
>>> baffle_correction_Bell(0.82)
1.1258554691854046
```

ht.conv_tube_bank.baffle_leakage_Bell(Ssb, Stb, Sm, method='spline')

Calculate the baffle leakage factor \( J_L \) which accounts for leakage between each baffle. Cubic spline interpolation is the default method of retrieving a value from the graph, which was digitized with Engauge-Digitizer.

The Heat Exchanger Design Handbook [4], [5] provides a curve fit as well. This method is not recommended, but can be used via the method “HEDH”.

\[ J_L = 0.44(1 - r_s) + [1 - 0.44(1 - r_s)] \exp(-2.2 r_{lm}) \]

\[ r_s = \frac{S_{sb}}{S_{sb} + S_{tb}} \]

\[ r_{lm} = \frac{S_{sb} + S_{tb}}{S_m} \]
Parameters

- **Ssb** [float] Shell to baffle leakage area, \([\text{m}^2]\)
- **Stb** [float] Total baffle leakage area, \([\text{m}^2]\)
- **Sm** [float] Crossflow area, \([\text{m}^2]\)
- **method** [str, optional] One of ‘spline’, or ‘HEDH’

Returns

- **Jl** [float] Baffle leakage factor in the Bell-Delaware method, [-]

Notes

Takes ~5 us per call. If the \(x\) parameter is larger than 0.743614, it is clipped to it.

The HEDH curve fits are rather poor and only 6x faster to evaluate. The HEDH example in [6]’s spreadsheet has an error and uses 0.044 instead of 0.44 in the equation.

References

[1], [2], [3], [4], [5], [6]

Examples

```python
>>> baffle_leakage_Bell(1, 3, 8)
0.5906621282470395
>>> baffle_leakage_Bell(1, 3, 8, 'HEDH')
0.5530236260777133
```

ht.conv_tube_bank.bundle_bypassing_Bell(bypass_area_fraction, seal_strips, crossflow_rows, laminar=False, method='spline')

Calculate the bundle bypassing effect \(J_b\) according to the Bell-Delaware method for heat exchanger design. Cubic spline interpolation is the default method of retrieving a value from the graph, which was digitized with Engauge-Digitizer.

The Heat Exchanger Design Handbook [4], [5] provides a curve fit as well. This method is not recommended, but can be used via the method “HEDH”:

\[
J_b = \exp\left[-1.25F_{slp}(1 - 2r_{ss}^{1/3})\right]
\]

For laminar flows, replace 1.25 with 1.35.

Parameters

- **bypass_area_fraction** [float] Fraction of the crossflow area which is not blocked by a baffle or anything else and available for bypassing, [-]
- **seal_strips** [int] Number of seal strips per side of a baffle added to prevent bypassing, [-]
- **crossflow_rows** [int] The number of tube rows in the crossflow of the baffle, [-]
- **laminar** [bool] Whether to use the turbulent correction values or the laminar ones; the Bell-Delaware method uses a Re criteria of 100 for this, [-]
- **method** [str, optional] One of ‘spline’, or ‘HEDH’

Returns
**Jb** [float] Bundle bypassing effect correction factor in the Bell-Delaware method. [-]

**Notes**

Takes ~5 us per call. If the *bypass_area_fraction* parameter is larger than 0.695, it is clipped to it.

**References**

[1], [2], [3]

**Examples**

```python
>>> bundle_bypassing_Bell(0.5, 5, 25)
0.8469611760884599
```

```python
>>> bundle_bypassing_Bell(0.5, 5, 25, method='HEDH')
0.8483210970579099
```

**ht.conv_tube_bank.unequal_baffle_spacing_Bell**

```python
ht.conv_tube_bank.unequal_baffle_spacing_Bell(baffles, baffle_spacing, baffle_spacing_in=None, baffle_spacing_out=None, laminar=False)
```

Calculate the correction factor for unequal baffle spacing *Js*, which accounts for higher velocity of fluid flow and greater heat transfer coefficients when the in and/or out baffle spacing is less than the standard spacing.

\[
J_s = \frac{(n_b - 1) + (B_{in}/B)^{(1-n_b)} + (B_{out}/B)^{(1-n_b)}}{(n_b - 1) + (B_{in}/B) + (B_{out}/B)}
\]

**Parameters**

- **baffles** [int] Number of baffles, [-]
- **baffle_spacing** [float] Average spacing between one end of one baffle to the start of the next baffle for non-exit baffles, [m]
- **baffle_spacing_in** [float, optional] Spacing between entrance to first baffle, [m]
- **baffle_spacing_out** [float, optional] Spacing between last baffle and exit, [m]
- **laminar** [bool, optional] Whether to use the turbulent exponent or the laminar one; the Bell-Delaware method uses a Re criteria of 100 for this, [-]

**Returns**

- **Js** [float] Unequal baffle spacing correction factor, [-]

**References**

[1], [2], [3], [4], [5]

**Examples**

```python
>>> unequal_baffle_spacing_Bell(16, .1, .15, 0.15)
0.9640087802805195
```
ht.conv_tube_bank.laminar_correction_Bell(Re, total_row_passes)

Calculate the correction factor for adverse temperature gradient built up in laminar flow $J_r$.

This correction begins at $Re = 100$, and is interpolated between the value of the formula until $Re = 20$, when it is the value of the formula. It is 1 for $Re > 100$. The value of the formula is not allowed to be less than 0.4.

$$J_r^* = \left( \frac{10}{N_{row,\text{passes,\text{tot}}}} \right)^{0.18}$$

Parameters

- **Re** [float] Shell Reynolds number in the Bell-Delaware method, [-]
- **total_row_passes** [int] The total number of rows passed by the fluid, including those in windows and counting repeat passes of tube rows, [-]

Returns

- **$J_r$** [float] Correction factor for adverse temperature gradient built up in laminar flow, [-]

Notes

[5] incorrectly uses the number of tube rows per crossflow section, not total.

References

[1], [2], [3], [4], [5]

Examples

```python
>>> laminar_correction_Bell(30, 80)
0.7267995454361379
```

2.16 Non boiling and non condensing two-phase heat transfer (ht.conv_two_phase)

ht.conv_two_phase.Davis_David(m, x, D, rhol, rhog, Cpl, kl, mul)

Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

$$h_{TP,T} = 0.060 \left( \frac{\rho_L}{\rho_G} \right)^{0.28} \left( \frac{DG_{TP,x}}{\mu_L} \right)^{0.87} \left( \frac{C_p,L\mu_L}{k_L} \right)^{0.4}$$

Parameters

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific tube interval [-]
- **D** [float] Diameter of the tube [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **Cpl** [float] Constant-pressure heat capacity of liquid [J/kg/K]
Heat Transfer Documentation, Release 0.1.56

```
kl [float] Thermal conductivity of liquid [W/m/K]

mul [float] Viscosity of liquid [Pa*s]

Returns

h [float] Heat transfer coefficient [W/m^2/K]

Notes

Developed for both vertical and horizontal flow, and flow patterns of annular or mist annular flow. Steam-water and air-water were the only considered fluid combinations. Quality ranged from 0.1 to 1 in their data. [1] claimed an AAE of 17%.

References

[1], [2]

Examples

```python
davis_david(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, Cpl=2300, kl=.6, ...
mul=1E-3)
```

```
ht.conv_two_phase.Elamvaluthi_Srinivas(m, x, D, rhol, rhog, Cpl, kl, mug, mu_b, mu_w=None)
```

Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

\[
\frac{h_{TP}D}{k_L} = 0.5 \left( \frac{\mu_G}{\mu_L} \right)^{0.25} Re_M^{0.7} Pr_L^{1/3} \left( \frac{\mu_b}{\mu_w} \right)^{0.14}
\]

\[
Re_M = \frac{DV_L\rho_L}{\mu_L} + \frac{DV_g\rho_g}{\mu_g}
\]

Parameters

m [float] Mass flow rate [kg/s]

x [float] Quality at the specific tube interval [-]

D [float] Diameter of the tube [m]

rhol [float] Density of the liquid [kg/m^3]

rhog [float] Density of the gas [kg/m^3]

Cpl [float] Constant-pressure heat capacity of liquid [J/kg/K]

kl [float] Thermal conductivity of liquid [W/m/K]

mug [float] Viscosity of gas [Pa*s]

mu_b [float] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]

mu_w [float, optional] Viscosity of liquid at wall temperature [Pa*s]

Returns

h [float] Heat transfer coefficient [W/m^2/K]
Notes

If the viscosity at the wall temperature is not given, the liquid viscosity correction is not applied.

Developed for vertical flow, and flow patterns of bubbly and slug. Gas/liquid superficial velocity ratios from 0.3 to 4.6, liquid mass fluxes from 200 to 1600 kg/m²s, and the fluids tested were air-water and air-aqueous glycerine solutions. The tube inner diameter was 1 cm, and the L/D ratio was 86.

References

[1], [2]

Examples

```python
>>> Elamvaluthi_Srinivas(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, Cpl=2300,
...   kl=.6, mug=1E-5, mu_b=1E-3, mu_w=1.2E-3)
3901.2134471578584
```

Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

\[
Re_M = \frac{DV_l \rho_l}{\mu_l} + \frac{DV_g \rho_g}{\mu_g}
\]

For the air-water system:

\[
\frac{h_{TPD}}{k_L} = 0.029 Re_M^{0.87} Pr_l^{1/3} (\mu_b/\mu_w)^{0.14}
\]

For gas/air-oil systems (default):

\[
\frac{h_{TPD}}{k_L} = 2.6 Re_M^{0.39} Pr_l^{1/3} (\mu_b/\mu_w)^{0.14}
\]

Parameters

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific tube interval [-]
- **D** [float] Diameter of the tube [m]
- **rhol** [float] Density of the liquid [kg/m³]
- **rhog** [float] Density of the gas [kg/m³]
- **Cpl** [float] Constant-pressure heat capacity of liquid [J/kg/K]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **mug** [float] Viscosity of gas [Pa*s]
- **mu_b** [float] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- **mu_w** [float, optional] Viscosity of liquid at wall temperature [Pa*s]
- **water** [bool, optional] Whether to use the water-air correlation or the gas/air-oil correlation

Returns

- **h** [float] Heat transfer coefficient [W/m²K]
Notes

If the viscosity at the wall temperature is not given, the liquid viscosity correction is not applied.

Developed for vertical pipes, with superficial velocity ratios of 0.6-250. Tested fluids were air-water, and gas/air-oil.

References

[1], [2]

Examples

```python
>>> Groothuis_Hendal(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, Cpl=2300, kl=.6,
...    mug=1E-5, mu_b=1E-3, mu_w=1.2E-3)
1192.9543445455754
```

ht.conv_two_phase. Hughmark (m, x, alpha, D, L, Cpl, kl, mu_b=None, mu_w=None)

Calculates the two-phase non-boiling laminar heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

\[
\frac{h_{TP}D}{k_l} = 1.75(1 - \alpha)^{-0.5} \left( \frac{m_l C_{p,l}}{(1 - \alpha) k_l L} \right)^{1/3} \left( \frac{\mu_b}{\mu_w} \right)^{0.14}
\]

Parameters

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific tube interval []
- **alpha** [float] Void fraction in the tube, []
- **D** [float] Diameter of the tube [m]
- **L** [float] Length of the tube, [m]
- **Cpl** [float] Constant-pressure heat capacity of liquid [J/kg/K]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **mu_b** [float] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- **mu_w** [float, optional] Viscosity of liquid at wall temperature [Pa*s]

Returns

- **h** [float] Heat transfer coefficient [W/m^2/K]

Notes

This model is based on a laminar entry length correlation - for a sufficiently long tube, this will predict unrealistically low heat transfer coefficients.

If the viscosity at the wall temperature is not given, the liquid viscosity correction is not applied.

Developed for horizontal pipes in laminar slug flow. Data consisted of the systems air-water, air-SAE 10 oil, gas-oil, air-diethylene glycol, and air-aqueous glycerine.
References

[1], [2]

Examples

```python
>>> Hughmark(m=1, x=.9, alpha=.9, D=.3, L=.5, Cpl=2300, kl=0.6, mu_b=1E-3,
... mu_w=1.2E-3)
212.7411636127175
```

ht.conv_two_phase.Knott(m, x, D, rhol, rhog, Cpl=None, kl=None, mu_b=None, mu_w=None,
L=None, hl=None)
Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

Either a specified \( h_l \) is required, or \( Cpl, kl, mu_b, mu_w \) and \( L \) are required to calculate \( h_l \).

\[
\frac{h_{TP}}{h_l} = \left( 1 + \frac{V_{gs}}{V_{ls}} \right)^{1/3}
\]

Parameters

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific tube interval [-]
- **D** [float] Diameter of the tube [m]
- **rhol** [float] Density of the liquid [kg/m^3]
- **rhog** [float] Density of the gas [kg/m^3]
- **Cpl** [float, optional] Constant-pressure heat capacity of liquid [J/kg/K]
- **kl** [float, optional] Thermal conductivity of liquid [W/m/K]
- **mu_b** [float, optional] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- **mu_w** [float, optional] Viscosity of liquid at wall temperature [Pa*s]
- **L** [float, optional] Length of the tube [m]
- **hl** [float, optional] Liquid-phase heat transfer coefficient as described below, [W/m^2/K]

Returns

- **h** [float] Heat transfer coefficient [W/m^2/K]

Notes

The liquid-only heat transfer coefficient will be calculated with the *laminar_entry_Seider_Tate* correlation, should it not be provided as an input. Many of the arguments to this function are optional and are only used if \( h_l \) is not provided.

\( h_l \) should be calculated with a velocity equal to that determined with a combined volumetric flow of both the liquid and the gas. All other parameters used in calculating the heat transfer coefficient are those of the liquid. If the viscosity at the wall temperature is not given, the liquid viscosity correction in *laminar_entry_Seider_Tate* is not applied.
References

[1], [2]

Examples

```python
Knot(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, Cpl=2300, kl=.6, mu_b=1E-3,
... mu_w=1.2E-3, L=4)
```

4225.536758045839

ht.conv_two_phase.Kudirka_Grosh_McFadden(m, x, D, rhol, rhog, Cpl, kl, mug, mu_b, mu_w=None)

Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

\[
N_u = \frac{h_{TP} D}{k_l} = 125 \left(\frac{V_{gs}}{V_{ls}}\right)^{0.125} \left(\frac{\mu_g}{\mu_l}\right)^{0.6} Re_{ls}^{0.25} Pr_l^{1/3} \left(\frac{\mu_b}{\mu_w}\right)^{0.14}
\]

Parameters

- m [float] Mass flow rate [kg/s]
- x [float] Quality at the specific tube interval [-]
- D [float] Diameter of the tube [m]
- rho_l [float] Density of the liquid [kg/m^3]
- rho_g [float] Density of the gas [kg/m^3]
- Cpl [float] Constant-pressure heat capacity of liquid [J/kg/K]
- kl [float] Thermal conductivity of liquid [W/m/K]
- mug [float] Viscosity of gas [Pa*s]
- mu_b [float] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- mu_w [float, optional] Viscosity of liquid at wall temperature [Pa*s]

Returns

- h [float] Heat transfer coefficient [W/m^2/K]

Notes

If the viscosity at the wall temperature is not given, the liquid viscosity correction is not applied.

Developed for air-water and air-ethylene glycol systems with a L/D of 17.6 and at low gas-liquid ratios. The flow regimes studied were bubble, slug, and froth flow.

References

[1], [2]

Examples
ht.conv_two_phase.Martin_Sims (m, x, D, rhol, rhog, hl=None, Cpl=None, kl=None, mu_b=None, mu_w=None, L=None)

Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

\[ \frac{h_{TP}}{h_l} = 1 + 0.64 \sqrt{\frac{V_g}{V_l}} \]

Parameters

- m [float] Mass flow rate [kg/s]
- x [float] Quality at the specific tube interval []
- D [float] Diameter of the tube [m]
- rhol [float] Density of the liquid [kg/m^3]
- rhog [float] Density of the gas [kg/m^3]
- hl [float, optional] Liquid-phase heat transfer coefficient as described below, [W/m^2/K]
- Cpl [float, optional] Constant-pressure heat capacity of liquid [J/kg/K]
- kl [float, optional] Thermal conductivity of liquid [W/m/K]
- mu_b [float, optional] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- mu_w [float, optional] Viscosity of liquid at wall temperature [Pa*s]
- L [float, optional] Length of the tube [m]

Returns

- h [float] Heat transfer coefficient [W/m^2/K]

Notes

No specific suggestion for how to calculate the liquid-phase heat transfer coefficient is given in [1]; [2] suggests to use the same procedure as in Knott.

References

[1], [2]

Examples

```python
>>> Kudirka_Grosh_McFadden(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, Cpl=2300, kl=.6, mub=1E-3, mug=1E-5, mu_w=1.2E-3)
303.9941255903587
```

```python
>>> Martin_Sims(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, hl=141.2)
5563.280000000001
```
ht.conv_two_phase.<strong>Ravipudi_Godbold</strong>(m, x, D, rhol, rhog, Cpl, kl, mug, mu_b, mu_w=None)
Calculates the two-phase non-boiling heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

\[
Nu = \frac{h_{TP} D}{k_l} = 0.56 \left( \frac{V_g}{V_l} \right)^{0.3} \left( \frac{\mu_g}{\mu_l} \right)^{0.2} \left( Re_{l,s}^{0.6} Pr_l^{1/3} \left( \frac{\mu_b}{\mu_w} \right)^{0.14} \left( \frac{\rho_g}{\rho_l} \right) \right)
\]

**Parameters**

- m [float] Mass flow rate [kg/s]
- x [float] Quality at the specific tube interval [-]
- D [float] Diameter of the tube [m]
- rhol [float] Density of the liquid [kg/m^3]
- rhog [float] Density of the gas [kg/m^3]
- Cpl [float] Constant-pressure heat capacity of liquid [J/kg/K]
- kl [float] Thermal conductivity of liquid [W/m/K]
- mug [float] Viscosity of gas [Pa*s]
- mu_b [float] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- mu_w [float, optional] Viscosity of liquid at wall temperature [Pa*s]

**Returns**

- h [float] Heat transfer coefficient [W/m^2/K]

**Notes**

If the viscosity at the wall temperature is not given, the liquid viscosity correction is not applied.

Developed with a vertical pipe, superficial gas/liquid velocity ratios of 1-90, in the froth regime, and for fluid mixtures of air and water, toluene, benzene, and methanol.

**References**

[1], [2]

**Examples**

```python
>>> Ravipudi_Godbold(m=1, x=.9, D=.3, rhol=1000, rhog=2.5, Cpl=2300, kl=.6, 
     mug=1E-5, mu_b=1E-3, mu_w=1.2E-3)
299.3796286459285
```

ht.conv_two_phase.<strong>Aggour</strong>(m, x, alpha, D, rhol, Cpl, kl, mu_b, mu_w=None, L=None, turbulent=None)
Calculates the two-phase non-boiling laminar heat transfer coefficient of a liquid and gas flowing inside a tube of any inclination, as in [1] and reviewed in [2].

Laminar for Rel <= 2000:

\[
h_{TP} = 1.615 \frac{k_l}{D} \left( \frac{Re_{l,s} D}{L} \right)^{1/3} \left( \frac{\mu_b}{\mu_w} \right)^{0.14}
\]

2.16. Non boiling and non condensing two-phase heat transfer (ht.conv_two_phase)
Turbulent for $\text{Rel} > 2000$:

$$h_{TP} = 0.0155 \frac{k_l}{D} P r_l^{0.5} R e_l^{0.83}$$

$$R e_l = \frac{\rho_l v_l D}{\mu_l}$$

$$V_l = \frac{V_{ls}}{1 - \alpha}$$

**Parameters**

- **m** [float] Mass flow rate [kg/s]
- **x** [float] Quality at the specific tube interval [-]
- **alpha** [float] Void fraction in the tube, [-]
- **D** [float] Diameter of the tube [m]
- **rhol** [float] Density of the liquid [kg/m$^3$]
- **Cpl** [float] Constant-pressure heat capacity of liquid [J/kg/K]
- **kl** [float] Thermal conductivity of liquid [W/m/K]
- **mu_b** [float] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- **mu_w** [float, optional] Viscosity of liquid at wall temperature [Pa*s]
- **L** [float, optional] Length of the tube, [m]
- **turbulent** [bool or None, optional] Whether or not to force the correlation to return the turbulent result; will return the laminar regime if False

**Returns**

- **h** [float] Heat transfer coefficient [W/m$^2$/K]

**Notes**

Developed with mixtures of air-water, helium-water, and freon-12-water and vertical tests. Studied flow patterns were bubbly, slug, annular, bubbly-slug, and slug-annular regimes. Superficial velocity ratios ranged from 0.02 to 470.

A viscosity correction is only suggested for the laminar regime. If the viscosity at the wall temperature is not given, the liquid viscosity correction is not applied.

**References**

[1], [2]

**Examples**

```python
>>> Aggour(m=1, x=.9, D=.3, alpha=.9, rhol=1000, Cpl=2300, kl=.6, mu_b=1E-3)
420.9347146885667
```

**ht.conv_two_phase.h_two_phase**

Calculates the two-phase non-boiling laminar heat transfer coefficient of a liquid and gas flowing inside a tube according to the specified method. Nine methods are available.
Parameters

- \( m \) [float] Mass flow rate [kg/s]
- \( x \) [float] Quality at the specific tube interval [-]
- \( D \) [float] Diameter of the tube [m]
- \( C_{pl} \) [float] Constant-pressure heat capacity of liquid [J/kg/K]
- \( k_l \) [float] Thermal conductivity of liquid [W/m/K]
- \( \rho_{l} \) [float, optional] Density of the liquid [kg/m^3]
- \( \rho_{g} \) [float, optional] Density of the gas [kg/m^3]
- \( \mu_{l} \) [float, optional] Viscosity of liquid [Pa*s]
- \( \mu_b \) [float, optional] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
- \( \mu_w \) [float, optional] Viscosity of liquid at wall temperature [Pa*s]
- \( \mu_g \) [float, optional] Viscosity of gas [Pa*s]
- \( L \) [float, optional] Length of the tube, [m]
- \( \alpha \) [float, optional] Void fraction in the tube, [-]

Returns

- \( h \) [float] Heat transfer coefficient [W/m^2/K]

Other Parameters

- \( \text{method} \) [string, optional] A string of the function name to use, as in the dictionary \texttt{conv_two_phase_methods}.

Notes

It is difficult to compare correlations.

Examples

```python
>>> h_two_phase(m=1, x=.9, D=.3, alpha=.9, rho_l=1000, C_{pl}=2300, k_l=.6, mu_b=1E-3, mu_w=1.2E-3, L=5, method='Aggour')
420.9347146885667
```

\texttt{ht.conv_two_phase.h_two_phase_methods} \( m, x, D, C_{pl}, k_l, \rho_{l}=\text{None}, \rho_{g}=\text{None}, \mu_{l}=\text{None}, \mu_{b}=\text{None}, \mu_{w}=\text{None}, \mu_{g}=\text{None}, L=\text{None}, \alpha=\text{None}, \text{check_ranges}=\text{True} \)

Returns a list of correlation names for the case of two-phase non-boiling liquid-gas laminar flow heat transfer inside a tube.

Parameters

- \( m \) [float] Mass flow rate [kg/s]
- \( x \) [float] Quality at the specific tube interval [-]
- \( D \) [float] Diameter of the tube [m]
- \( C_{pl} \) [float] Constant-pressure heat capacity of liquid [J/kg/K]
kl [float] Thermal conductivity of liquid [W/m/K]
rhol [float, optional] Density of the liquid [kg/m^3]
rhog [float, optional] Density of the gas [kg/m^3]
mul [float, optional] Viscosity of liquid [Pa*s]
mu_b [float, optional] Viscosity of liquid at bulk conditions (average of inlet/outlet temperature) [Pa*s]
mu_w [float, optional] Viscosity of liquid at wall temperature [Pa*s]
mug [float, optional] Viscosity of gas [Pa*s]
L [float, optional] Length of the tube, [m]
alpha [float, optional] Void fraction in the tube, [-]
check_ranges [bool, optional] Whether or not to return only correlations suitable for the provided data, [-]

Returns

h [float] Heat transfer coefficient [W/m^2/K]

Notes

A review of the correlations for which has the best performance has not been performed.

Examples

```python
>>> h_two_phase_methods(m=1, x=.9, D=.3, alpha=.9, rhol=1000, Cpl=2300, kl=.6, mu_b=1E-3, mu_w=1.2E-3, L=5)
'Aggour'
```

2.17 Miscellaneous utilities (ht.core)

ht.core.LMTD (Thi, Tho, Tci, Tco, counterflow=True)

Returns the log-mean temperature difference of an ideal counterflow or co-current heat exchanger.

\[
\Delta T_{LMTD} = \frac{\Delta T_1 - \Delta T_2}{\ln(\Delta T_1 / \Delta T_2)}
\]

For countercurrent:

\[
\Delta T_1 = T_{h,i} - T_{c,o}
\]

\[
\Delta T_2 = T_{h,o} - T_{c,i}
\]

Parallel Flow Only:

\[
\Delta T_1 = T_{h,i} - T_{c,i}
\]

\[
\Delta T_2 = T_{h,o} - T_{c,o}
\]

Parameters

Thi [float] Inlet temperature of hot fluid, [K]
Tho [float] Outlet temperature of hot fluid, [K]
**Tci** [float] Inlet temperature of cold fluid, [K]

**Tco** [float] Outlet temperature of cold fluid, [K]

**counterflow** [bool, optional] Whether the exchanger is counterflow or co-current

**Returns**

**LMTD** [float] Log-mean temperature difference [K]

**Notes**

Any consistent set of units produces a consistent output.

For the case where the temperature difference is the same in counterflow, the arithmetic mean difference (either difference in that case) is the correct result following evaluation of the limit.

For the same problem with the co-current case, the limit evaluates to a temperature difference of zero.

**Examples**

Example 11.1 in [1].

```python
>>> LMTD(100., 60., 30., 40.2)
43.200409294131525
>>> LMTD(100., 60., 30., 40.2, counterflow=False)
39.75251118049003
>>> LMTD(100., 60., 20., 60)
40.0
>>> LMTD(100., 60., 20., 60, counterflow=False)
0.0
```

**ht.core.wall_factor** *(mu=None, mu_wall=None, Pr=None, Pr_wall=None, T=None, T_wall=None, mu_heating_coeff=0.11, mu_cooling_coeff=0.25, Pr_heating_coeff=0.11, Pr_cooling_coeff=0.25, T_heating_coeff=0.11, T_cooling_coeff=0.25, property_option='Prandtl')*

Computes the wall correction factor for heat transfer, mass transfer, or momentum transfer between a fluid and a wall. Utility function; the coefficients for the phenomenon must be provided to this method. The default coefficients are for heat transfer of a turbulent liquid.

The general formula is as follows; substitute the property desired and the phenomenon desired into the equation for things other than heat transfer.

\[
\frac{N_u}{N_u_{\text{constant properties}}} = \left(\frac{\mu}{\mu_{\text{wall}}}\right)^n
\]

**Parameters**

**mu** [float, optional] Viscosity of flowing fluid away from the surface, [Pa*s]

**mu_wall** [float, optional] Viscosity of the fluid at the wall, [Pa*s]

**Pr** [float, optional] Prandtl number of flowing fluid away from the surface, [-]

**Pr_wall** [float, optional] Prandtl number of the fluid at the wall, [-]
T  [float, optional] Temperature of flowing fluid away from the surface, [K]
T_wall  [float, optional] Temperature of the fluid at the wall, [K]
mu_heating_coeff  [float, optional] Coefficient for viscosity - surface providing heating, [-]
mu_cooling_coeff  [float, optional] Coefficient for viscosity - surface providing cooling, [-]
Pr_heating_coeff  [float, optional] Coefficient for Prandtl number - surface providing heating, [-]
Pr_cooling_coeff  [float, optional] Coefficient for Prandtl number - surface providing cooling, [-]
T_heating_coeff  [float, optional] Coefficient for temperature - surface providing heating, [-]
T_cooling_coeff  [float, optional] Coefficient for temperature - surface providing cooling, [-]
property_option  [str, optional] Which property to use for computing the correction factor; one
of ‘Viscosity’, ‘Prandtl’, or ‘Temperature’.

Returns
factor  [float] Correction factor for heat transfer; to be multiplied by the Nusselt number or heat
transfer coefficient or friction factor or pressure drop to obtain the actual result, [-]

Examples

```python
>>> wall_factor(mu=8E-4, mu_wall=3E-4, Pr=1.2, Pr_wall=1.1, T=300, ...
... T_wall=350, property_option='Prandtl')
1.0096172023817749
```

ht.core.is_heating_property (prop, prop_wall)
Checks whether or not a fluid side is being heated or cooled, from a property of the fluid at the wall and the bulk
temperature. Returns True for heating the bulk fluid, and False for cooling the bulk fluid.

Parameters

prop  [float] Viscosity (or Prandtl number) of flowing fluid away from the heat transfer surface,
[Pa*s]
prop_wall  [float] Viscosity (or Prandtl number) of the fluid at the wall, [Pa*s]

Returns

is_heating  [bool] Whether or not the flow is being heated up by the wall, [-]

Examples

```python
>>> is_heating_property(1E-3, 1.2E-3)
False
```

ht.core.is_heating_temperature (T, T_wall)
Checks whether or not a fluid side is being heated or cooled, from the temperature of the wall and the bulk
temperature. Returns True for heating the bulk fluid, and False for cooling the bulk fluid.

Parameters

T  [float] Temperature of flowing fluid away from the heat transfer surface, [K]
T_wall  [float] Temperature of the fluid at the wall, [K]
Returns

is_heating [bool] Whether or not the flow is being heated up by the wall, [-]

Examples

```python
>>> is_heating_temperature(298.15, 350)
True
```

ht.core.wall_factor_fd(mu, mu_wall, turbulent=True, liquid=False)

Computes the wall correction factor for pressure drop due to friction between a fluid and a wall. These coefficients were derived for internal flow inside a pipe, but can be used elsewhere where appropriate data is missing.

$$\frac{f_d}{f_{d,\text{constant properties}}} = \left( \frac{\mu}{\mu_{\text{wall}}} \right)^n$$

Parameters

- mu [float] Viscosity (or Prandtl number) of flowing fluid away from the wall, [Pa*s]
- mu_wall [float] Viscosity (or Prandtl number) of the fluid at the wall, [Pa*s]
- turbulent [bool] Whether or not to use the turbulent coefficient, [-]
- liquid [bool] Whether or not to use the liquid phase coefficient; otherwise the gas coefficient is used, [-]

Returns

factor [float] Correction factor for pressure loss; to be multiplied by the friction factor, or pressure drop to obtain the actual result, [-]

Notes

The exponents are determined as follows:

<table>
<thead>
<tr>
<th>Regime</th>
<th>Phase</th>
<th>Heating</th>
<th>Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turbulent</td>
<td>Liquid</td>
<td>-0.25</td>
<td>-0.25</td>
</tr>
<tr>
<td>Turbulent</td>
<td>Gas</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Laminar</td>
<td>Liquid</td>
<td>-0.58</td>
<td>-0.5</td>
</tr>
<tr>
<td>Laminar</td>
<td>Gas</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

References

[1]

Examples

```python
>>> wall_factor_fd(mu=8E-4, mu_wall=3E-4, turbulent=True, liquid=True)
0.7825422900366437
```

ht.core.wall_factor_Nu(mu, mu_wall, turbulent=True, liquid=False)

Computes the wall correction factor for heat transfer between a fluid and a wall. These coefficients were derived
Heat Transfer Documentation, Release 0.1.56

for internal flow inside a pipe, but can be used elsewhere where appropriate data is missing. It is also useful to compare these results with the coefficients used in various heat transfer coefficients.

\[
\frac{N_u}{N_{u_{\text{constant properties}}}} = \left(\frac{\mu}{\mu_{\text{wall}}}\right)^n
\]

Parameters

- mu [float] Viscosity (or Prandtl number) of flowing fluid away from the heat transfer surface, [Pa*s]
- mu_wall [float] Viscosity (or Prandtl number) of the fluid at the wall, [Pa*s]
- turbulent [bool] Whether or not to use the turbulent coefficient, [-]
- liquid [bool] Whether or not to use the liquid phase coefficient; otherwise the gas coefficient is used, [-]

Returns

- factor [float] Correction factor for heat transfer; to be multiplied by the Nusselt number, or heat transfer coefficient to obtain the actual result, [-]

Notes

The exponents are determined as follows:

<table>
<thead>
<tr>
<th>Regime</th>
<th>Phase</th>
<th>Heating</th>
<th>Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turbulent</td>
<td>Liquid</td>
<td>0.11</td>
<td>0.25</td>
</tr>
<tr>
<td>Turbulent</td>
<td>Gas</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>Laminar</td>
<td>Liquid</td>
<td>0.14</td>
<td>0.14</td>
</tr>
<tr>
<td>Laminar</td>
<td>Gas</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

References

[1]

Examples

```python
>>> wall_factor_Nu(mu=8E-4, mu_wall=3E-4, turbulent=True, liquid=True)
1.1139265634480144

>>> wall_factor_Nu(mu=8E-4, mu_wall=3E-4, turbulent=False, liquid=True)
1.147190712947014

>>> wall_factor_Nu(mu=1.5E-5, mu_wall=1.3E-5, turbulent=True, liquid=False)
1.0741723110591495

>>> wall_factor_Nu(mu=1.5E-5, mu_wall=1.3E-5, turbulent=False, liquid=False)
1.0
```

ht.core.fin_efficiency_Kern_Kraus(Do, D_fin, t_fin, k_fin, h)

Returns the efficiency eta_f of a circular fin of constant thickness attached to a circular tube, based on the tube
diameter Do, fin diameter D_fin, fin thickness t_fin, fin thermal conductivity k_fin, and heat transfer coefficient h.

\[
\eta_f = \frac{2r_o}{m(r_e^2 - r_o^2)} \left[ \frac{I_1(mr_e)K_1(mr_o) - K_1(mr_e)I_1(mr_o)}{I_0(mr_o)K_1(mr_e) + I_1(mr_e)K_0(mr_o)} \right]
\]

\[
m = \sqrt{\frac{2h}{k_fin t_fin}}
\]

\[
r_e = 0.5D_fin
\]

\[
r_o = 0.5D_o
\]

Parameters

Do [float] Outer diameter of bare pipe (as if there were no fins), [m]

D_fin [float] Outer diameter of the fin, from the center of the tube to the edge of the fin, [m]

t_fin [float] Thickness of the fin (for constant thickness fins only), [m]

k_fin [float] Thermal conductivity of the fin, [W/m/K]

h [float] Heat transfer coefficient of the finned pipe, [W/K]

Returns

eta_fin [float] Fin efficiency [-]

Notes

I0, I1, K0 and K1 are modified Bessel functions of order 0 and 1, modified Bessel function of the second kind of order 0 and 1 respectively.

References

[1], [2], [3], [4]

Examples

```python
>>> fin_efficiency_Kern_Kraus(0.0254, 0.05715, 3.8E-4, 200, 58)
0.8412588620231153
```

2.18 Heat exchanger sizing and rating (ht.hx)

ht.hx.effectiveness_from_NTU(NTU, Cr, subtype='counterflow')

Returns the effectiveness of a heat exchanger at a specified heat capacity rate, number of transfer units, and configuration. The following configurations are supported:

- Counterflow (ex. double-pipe)
- Parallel (ex. double pipe inefficient configuration)
- Shell and tube exchangers with even numbers of tube passes, one or more shells in series
- Crossflow, single pass, fluids unmixed
• Crossflow, single pass, Cmax mixed, Cmin unmixed
• Crossflow, single pass, Cmin mixed, Cmax unmixed
• Boiler or condenser

These situations are normally not those which occur in real heat exchangers, but are useful for academic purposes. More complicated expressions are available for other methods. These equations are confirmed in [1], [2], and [3].

For parallel flow heat exchangers:
\[ \epsilon = 1 - \exp[-NTU(1 + C_r)] \]

For counterflow heat exchangers:
\[ \epsilon = \frac{1 - \exp[-NTU(1 - C_r)]}{1 - C_r \exp[-NTU(1 - C_r)]}, \quad C_r < 1 \]
\[ \epsilon = \frac{NTU}{1 + NTU}, \quad C_r = 1 \]

For TEMA E shell-and-tube heat exchangers with one shell pass, 2n tube passes:
\[ \epsilon_1 = 2 \left\{ 1 + C_r + \sqrt{1 + C_r^2} \times \frac{1 + \exp[-(NTU)_1 \sqrt{1 + C_r^2}]}{1 - \exp[-(NTU)_1 \sqrt{1 + C_r^2}]} \right\}^{-1} \]

For TEMA E shell-and-tube heat exchangers with more than one shell pass, 2n tube passes (this model assumes each exchanger has an equal share of the overall NTU or said more plainly, the same UA):
\[ \epsilon = \left[ \left( \frac{1 - \epsilon_1 C_r}{1 - \epsilon_1} \right)^2 - 1 \right] \left[ \left( \frac{1 - \epsilon_1 C_r}{1 - \epsilon_1} \right)^n - C_r \right]^{-1} \]

For cross-flow (single-pass) heat exchangers with both fluids unmixed, there is an approximate and an exact formula. The approximate one is:
\[ \epsilon = 1 - \exp \left( \frac{1}{C_r} \right) (NTU)^{0.22} \{ \exp \left[ C_r (NTU)^{0.78} \right] - 1 \} \]

The exact solution for crossflow (fluids unmixed) uses SciPy’s quad to perform an integral (there is no analytical integral solution available). \( I_0(v) \) is the modified Bessel function of the first kind. This formula was developed in [4].
\[ \epsilon = \frac{1}{C_r} - \frac{\exp(-C_r \cdot NTU)}{2(C_r NTU)^2} \int_0^{2NTU\sqrt{C_r}} \left( 1 + NTU - \frac{v^2}{4C_r NTU} \right) \exp \left( -\frac{v^2}{4C_r NTU} \right) v I_0(v) dv \]

For cross-flow (single-pass) heat exchangers with Cmax mixed, Cmin unmixed:
\[ \epsilon = \left( \frac{1}{C_r} \right) (1 - \exp \{-C_r[1 - \exp(-NTU)]\}) \]

For cross-flow (single-pass) heat exchangers with Cmin mixed, Cmax unmixed:
\[ \epsilon = 1 - \exp(-C_r^{-1}[1 - \exp(-C_r(NTU))]) \]

For cases where \( Cr = 0 \), as in an exchanger with latent heat exchange, flow arrangement does not matter:
\[ \epsilon = 1 - \exp(-NTU) \]
Parameters

- **NTU** [float] Thermal Number of Transfer Units [-]
- **Cr** [float] The heat capacity rate ratio, of the smaller fluid to the larger fluid, [-]

Returns

- **effectiveness** [float] The thermal effectiveness of the heat exchanger, [-]

Notes

Once the effectiveness of the exchanger has been calculated, the total heat transferred can be calculated according to the following formulas, depending on which stream temperatures are known:

If the inlet temperatures for both sides are known:

\[ Q = \epsilon C_{\text{min}} (T_{h,i} - T_{c,i}) \]

If the outlet temperatures for both sides are known:

\[ Q = \frac{\epsilon C_{\text{min}} (C_{\text{hot}} C_{\text{cold}} (T_{c,o} - T_{h,o}))}{\epsilon C_{\text{min}} (C_{\text{hot}} + C_{\text{cold}}) - (C_{\text{hot}} C_{\text{cold}})} \]

If the hot inlet and cold outlet are known:

\[ Q = \frac{\epsilon C_{\text{min}} C_c (T_{c,o} - T_{h,i})}{\epsilon C_{\text{min}} - C_c} \]

If the hot outlet stream and cold inlet stream are known:

\[ Q = \frac{\epsilon C_{\text{min}} C_h (T_{c,i} - T_{h,o})}{\epsilon C_{\text{min}} - C_h} \]

If the inlet and outlet conditions for a single side are known, the effectiveness wasn’t needed for it to be calculated. For completeness, the formulas are as follows:

\[ Q = C_c (T_{c,o} - T_{c,i}) = C_h (T_{h,i} - T_{h,o}) \]

There is also a term called \( Q_{\text{max}} \), which is the heat which would have been transferred if the effectiveness was 1. It is calculated as follows:

\[ Q_{\text{max}} = \frac{Q}{\text{effectiveness}} \]

References

[1], [2], [3], [4]

Examples

Worst case, parallel flow:
Crossflow, somewhat higher effectiveness:

\[
\text{effectiveness_from_NTU(NTU=5, Cr=0.7, subtype='crossflow')} = 0.844482179974855
\]

Counterflow, better than either crossflow or parallel flow:

\[
\text{effectiveness_from_NTU(NTU=5, Cr=0.7, subtype='counterflow')} = 0.9206703686051108
\]

One shell and tube heat exchanger gives worse performance than counterflow, but they are designed to be economical and compact which a counterflow exchanger would not be. As the number of shells approaches infinity, the counterflow result is obtained exactly.

\[
\text{effectiveness_from_NTU(NTU=5, Cr=0.7, subtype='S&T')} = 0.6834977044311439
\]

\[
\text{effectiveness_from_NTU(NTU=5, Cr=0.7, subtype='50S&T')} = 0.9205058702789254
\]

Overall case of rating an existing heat exchanger where a known flowrate of steam and oil are contacted in crossflow, with the steam side mixed (example 10-9 in [3]):

\[
\begin{align*}
U &= 275 \text{ } \# \text{ } \text{W/m}^2/\text{K} \\
A &= 10.82 \text{ } \# \text{ } \text{m}^2 \\
C_{p_oil} &= 1900 \text{ } \# \text{ } \text{J/kg/K} \\
C_{p_steam} &= 1860 \text{ } \# \text{ } \text{J/kg/K} \\
m_{steam} &= 5.2 \text{ } \# \text{ } \text{kg/s} \\
m_{oil} &= 0.725 \text{ } \# \text{ } \text{kg/s} \\
\text{Thi} &= 130 \text{ } \# \text{ } \text{°C} \\
\text{Tci} &= 15 \text{ } \# \text{ } \text{°C} \\
C_{min} &= \text{calc}_C_{min}(m_h=m_{steam}, m_c=m_{oil}, C_{ph}=C_{p_steam}, C_{pc}=C_{p_oil}) \\
C_{max} &= \text{calc}_C_{max}(m_h=m_{steam}, m_c=m_{oil}, C_{ph}=C_{p_steam}, C_{pc}=C_{p_oil}) \\
Cr &= \text{calc}_C_{r}(m_h=m_{steam}, m_c=m_{oil}, C_{ph}=C_{p_steam}, C_{pc}=C_{p_oil}) \\
NTU &= \text{NTU_from_UA}(U*A, C_{min}=C_{min}) \\
\text{eff} &= \text{effectiveness_from_NTU}(NTU=NTU, Cr=Cr, subtype='crossflow, mixed Cmax') \\
Q &= \text{eff} \ast C_{min} \ast (\text{Thi} - \text{Tci}) \\
\text{Tco} &= \text{Tci} + Q/(m_{oil} \ast C_{p_oil}) \\
\text{Tho} &= \text{Thi} - Q/(m_{steam} \ast C_{p_steam}) \\
\text{Cmin}, \text{Cmax}, \text{Cr} &= (1377.5, 9672.0, 0.14242142266335814) \\
\text{NTU, eff, Q} &= (2.160072595281307, 0.8312180361425988, 131675.32715043944) \\
\text{Tco, Tho} &= (110.59007415639887, 116.38592564614977)
\end{align*}
\]

Alternatively, if only the outlet temperatures had been known:

\[
\begin{align*}
\text{Tco} &= 110.59007415639887 \\
\text{Tho} &= 116.38592564614977 \\
\text{Cc, Ch} &= \text{Cmin, Cmax} \text{ } \# \text{ In this case but not always} \\
Q &= \text{eff} \ast \text{Cmin} \ast \text{Cc} \ast \text{Ch} \ast (\text{Tco} - \text{Tho})/(\text{eff} \ast \text{Cmin} \ast (\text{Cc} + \text{Ch}) - \text{Ch} \ast \text{Cc}) \\
\text{Thi} &= \text{Tho} + Q/\text{Ch} \\
\text{Tci} &= \text{Tco} - Q/\text{Cc}
\end{align*}
\]
ht.hx.NTU_from_effectiveness (effectiveness, Cr, subtype='counterflow')

Returns the Number of Transfer Units of a heat exchanger at a specified heat capacity rate, effectiveness, and configuration. The following configurations are supported:

- Counterflow (ex. double-pipe)
- Parallel (ex. double pipe inefficient configuration)
- Shell and tube exchangers with even numbers of tube passes, one or more shells in series (TEMA E (one pass shell) only)
- Crossflow, single pass, fluids unmixed
- Crossflow, single pass, Cmax mixed, Cmin unmixed
- Crossflow, single pass, Cmin mixed, Cmax unmixed
- Boiler or condenser

These situations are normally not those which occur in real heat exchangers, but are useful for academic purposes. More complicated expressions are available for other methods. These equations are confirmed in [1], [2], and [3].

For parallel flow heat exchangers:

\[
NTU = \frac{-\ln[1 - \epsilon(1 + C_r)]}{1 + C_r}
\]

For counterflow heat exchangers:

\[
NTU = \frac{1}{C_r - 1} \ln \left( \frac{\epsilon - 1}{\epsilon C_r - 1} \right)
\]

\[
NTU = \frac{\epsilon}{1 - \epsilon} \text{ if } C_r = 1
\]

For TEMA E shell-and-tube heat exchangers with one shell pass, 2n tube passes:

\[
(NTU)_1 = -(1 + C_r^2)^{-0.5} \ln \left( \frac{E - 1}{E + 1} \right)
\]

\[
E = \frac{2/\epsilon_1 - (1 + C_r)}{(1 + C_r^2)^{0.5}}
\]

For TEMA E shell-and-tube heat exchangers with more than one shell pass, 2n tube passes (this model assumes each exchanger has an equal share of the overall NTU or said more plainly, the same UA):

\[
\epsilon_1 = \frac{F - 1}{F - C_r}
\]

\[
F = \left( \frac{\epsilon C_r - 1}{\epsilon - 1} \right)^{1/n}
\]

\[
NTU = n(NTU)_1
\]

For cross-flow (single-pass) heat exchangers with both fluids unmixed, there is no analytical solution. However, the function is monotonically increasing, and a closed-form solver is implemented as 'crossflow approximate', guaranteed to solve for $10^{-7} < NTU < 10^5$. The exact solution for 'crossflow' uses the approximate solution's
initial guess as a starting point for Newton’s method. Some issues are noted at effectivenesses higher than 0.9 and very high NTUs, because the numerical integral term approaches 1 too quickly.

For cross-flow (single-pass) heat exchangers with Cmax mixed, Cmin unmixed:

\[
NTU = -\ln \left( 1 + \frac{1}{C_r} \ln(1 - \epsilon C_r) \right)
\]

For cross-flow (single-pass) heat exchangers with Cmin mixed, Cmax unmixed:

\[
NTU = -\frac{1}{C_r} \ln[C_r \ln(1 - \epsilon) + 1]
\]

For cases where \( C_r = 0 \), as in an exchanger with latent heat exchange, flow arrangement does not matter:

\[
NTU = -\ln(1 - \epsilon)
\]

**Parameters**

- **effectiveness** [float] The thermal effectiveness of the heat exchanger, [-]
- **Cr** [float] The heat capacity rate ratio, of the smaller fluid to the larger fluid, [-]

**Returns**

- **NTU** [float] Thermal Number of Transfer Units [-]

**Notes**

Unlike `ht.hx.effectiveness_from_NTU`, not all inputs can calculate the NTU - many exchanger types have effectiveness limits below 1 which depend on \( C_r \) and the number of shells in the case of heat exchangers. If an impossible input is given, an error will be raised and the maximum possible effectiveness will be printed.

```python
>>> NTU_from_effectiveness(.99, Cr=.7, subtype='5S&T') # doctest: +SKIP
Traceback (most recent call last):
  Exception: The specified effectiveness is not physically possible for this
  configuration; the maximum effectiveness possible is 0.974122977755.
```

**References**

[1], [2], [3]

**Examples**

Worst case, parallel flow:

```python
>>> NTU_from_effectiveness(effectiveness=0.5881156068417585, Cr=0.7, subtype='parallel')
5.000000000000012
```

Crossflow, somewhat higher effectiveness:
Counterflow, better than either crossflow or parallel flow:

Shell and tube exchangers:

Overall case of rating an existing heat exchanger where a known flowrate of steam and oil are contacted in crossflow, with the steam side mixed, known inlet and outlet temperatures, and unknown UA (based on example 10-8 in [3]):

```
>>> Cp_oil = 1900  # J/kg/K
>>> Cp_steam = 1860  # J/kg/K
>>> m_steam = 5.2  # kg/s
>>> m_oil = 1.45  # kg/s
>>> Thi = 130  # °C
>>> Tci = 15  # °C
>>> Tco = 85  # °C  # Design specification
>>> Q = Cp_oil*m_oil*(Tci-Tco)
>>> dTh = Q/(m_steam*Cp_steam)
>>> Tho = Thi + dTh
>>> Cmin = calc_Cmin(mh=m_steam, mc=m_oil, Cph=Cp_steam, Cpc=Cp_oil)
>>> Cmax = calc_Cmax(mh=m_steam, mc=m_oil, Cph=Cp_steam, Cpc=Cp_oil)
>>> Cr = calc_Cr(mh=m_steam, mc=m_oil, Cph=Cp_steam, Cpc=Cp_oil)
>>> effectiveness = -Q/Cmin/(Thi-Tci)
>>> NTU = NTU_from_effectiveness(effectiveness, Cr, subtype='crossflow, mixed Cmax →')
>>> UA = UA_from_NTU(NTU, Cmin)
>>> U = 200  # Assume this was calculated; would actually need to be obtained iteratively as U depends on the exchanger geometry
>>> A = UA/U
>>> Tho, Cmin, Cmax, Cr
(110.06100082712986, 2755.0, 9672.0, 0.2848428453267163)
```

ht.hx.calc_Cmin (mh, mc, Cph, Cpc)

Returns the heat capacity rate for the minimum stream having flows mh and mc, with averaged heat capacities Cph and Cpc.

\[
C_c = m_c C_{p,c} \\
C_h = m_h C_{p,h} \\
C_{min} = \min(C_c, C_h)
\]

Parameters
**Returns**

- **Cmin** [float] The heat capacity rate of the smaller fluid, [W/K]

**Notes**

Used with the effectiveness method for heat exchanger design. Technically, it doesn’t matter if the hot and cold streams are in the right order for the input, but it is easiest to use this function when the order is specified.

**References**

[1]

**Examples**

```python
>>> calc_Cmin(mh=22., mc=5.5, Cph=2200, Cpc=4400.)
24200.0
```

```python
def calc_Cmax(mh, mc, Cph, Cpc):
    # Returns the heat capacity rate for the maximum stream having flows mh and mc, with averaged heat capacities Cph and Cpc.

    Cc = mc*Cpc
    Ch = mh*Cph
    Cmax = max(Cc, Ch)
    return Cmax
```

**Parameters**

- **mh** [float] Mass flow rate of hot stream, [kg/s]
- **mc** [float] Mass flow rate of cold stream, [kg/s]
- **Cph** [float] Averaged heat capacity of hot stream, [J/kg/K]
- **Cpc** [float] Averaged heat capacity of cold stream, [J/kg/K]

**Returns**

- **Cmax** [float] The heat capacity rate of the larger fluid, [W/K]

**Notes**

Used with the effectiveness method for heat exchanger design. Technically, it doesn’t matter if the hot and cold streams are in the right order for the input, but it is easiest to use this function when the order is specified.
References

[1]

Examples

```python
>>> calc_Cmax(mh=22., mc=5.5, Cph=2200, Cpc=4400.)
48400.0
```

```
ht.hx.calc_Cr(mh, mc, Cph, Cpc)
```

Returns the heat capacity rate ratio for a heat exchanger having flows $mh$ and $mc$, with averaged heat capacities $Cph$ and $Cpc$.

$$C_r = C^* = \frac{C_{\text{min}}}{C_{\text{max}}}$$

Parameters

- **mh** [float] Mass flow rate of hot stream, [kg/s]
- **mc** [float] Mass flow rate of cold stream, [kg/s]
- **Cph** [float] Averaged heat capacity of hot stream, [J/kg/K]
- **Cpc** [float] Averaged heat capacity of cold stream, [J/kg/K]

Returns

- **Cr** [float] The heat capacity rate ratio, of the smaller fluid to the larger fluid, [W/K]

Notes

Used with the effectiveness method for heat exchanger design. Technically, it doesn’t matter if the hot and cold streams are in the right order for the input, but it is easiest to use this function when the order is specified.

References

[1]

Examples

```python
>>> calc_Cr(mh=22., mc=5.5, Cph=2200, Cpc=4400.)
0.5
```

```
ht.hx.Pp(x, y)
```

Basic helper calculator which accepts a transformed $R1$ and $NTU1$ as inputs for a common term used in the calculation of the $P$-$NTU$ method for plate exchangers.

Returns a value which is normally used in other calculations before the actual $P1$ is calculated.

$$P_p(x, y) = \frac{1 - \exp[-x(1 + y)]}{1 + y}$$

Parameters

- **x** [float] A modification of $NTU1$, the Thermal Number of Transfer Units [-]
ht.hx.Pc(x, y)

Basic helper calculator which accepts a transformed R1 and NTU1 as inputs for a common term used in the calculation of the P-NTU method for plate exchangers.

Returns a value which is normally used in other calculations before the actual P1 is calculated. Nominally used in counterflow calculations

\[
P_c(x, y) = \frac{1 - \exp[-x(1-y)]}{1 - y \exp[-x(1-y)]}
\]

Parameters

x [float] A modification of NTU1, the Thermal Number of Transfer Units [-]
y [float] A modification of R1, the thermal effectiveness [-]

Returns

z [float] Just another term in the calculation, [-]

Notes

Used with the P-NTU plate method for heat exchanger design. At y = -1, this function has a ZeroDivisionError but can be evaluated at the limit to be \( z = \frac{x}{1+x} \).
Examples

```python
>>> Pc(5, .7)
0.9206703686051108
```

ht.hx.NTU_from_UA(UA, Cmin)
Returns the Number of Transfer Units for a heat exchanger having $UA$, and with $Cmin$ heat capacity rate.

$$NTU = \frac{UA}{C_{min}}$$

Parameters

- **UA** [float] Combined Area-heat transfer coefficient term, [W/K]
- **Cmin** [float] The heat capacity rate of the smaller fluid, [W/K]

Returns

- **NTU** [float] Thermal Number of Transfer Units [-]

Notes

Used with the effectiveness method for heat exchanger design.

References

[1]

Examples

```python
>>> NTU_from_UA(4400., 22.)
200.0
```

ht.hx.UA_from_NTU(NTU, Cmin)
Returns the combined area-heat transfer term for a heat exchanger having a specified $NTU$, and with $Cmin$ heat capacity rate.

$$UA = NTU C_{min}$$

Parameters

- **NTU** [float] Thermal Number of Transfer Units [-]
- **Cmin** [float] The heat capacity rate of the smaller fluid, [W/K]

Returns

- **UA** [float] Combined area-heat transfer coefficient term, [W/K]

Notes

Used with the effectiveness method for heat exchanger design.
References

[1]

Examples

```
>>> UA_from_NTU(200., 22.)
4400.0
```

ht.hx.effectiveness_NTU_method(mh, mc, Cph, Cpc, subtype='counterflow', Thi=None, Tho=None, Tci=None, Tco=None, UA=None)

Wrapper for the various effectiveness-NTU method function calls, which can solve a heat exchanger. The heat capacities and mass flows of each stream and the type of the heat exchanger are always required. As additional inputs, one combination of the following inputs is required:

- Three of the four inlet and outlet stream temperatures.
- Temperatures for the cold outlet and hot inlet and UA
- Temperatures for the cold outlet and hot outlet and UA
- Temperatures for the cold inlet and hot inlet and UA
- Temperatures for the cold inlet and hot outlet and UA

Parameters

- **mh** [float] Mass flow rate of hot stream, [kg/s]
- **mc** [float] Mass flow rate of cold stream, [kg/s]
- **Cph** [float] Averaged heat capacity of hot stream, [J/kg/K]
- **Cpc** [float] Averaged heat capacity of cold stream, [J/kg/K]
- **subtype** [str, optional] The subtype of exchanger, one of ‘counterflow’, ‘parallel’, ‘crossflow’ ‘crossflow, mixed Cmin’, ‘crossflow, mixed Cmax’, ‘boiler’, ‘condenser’, ‘S&T’, or ‘nS&T’ where n is the number of shell and tube exchangers in a row
- **Thi** [float, optional] Inlet temperature of hot fluid, [K]
- **Tho** [float, optional] Outlet temperature of hot fluid, [K]
- **Tci** [float, optional] Inlet temperature of cold fluid, [K]
- **Tco** [float, optional] Outlet temperature of cold fluid, [K]
- **UA** [float, optional] Combined Area-heat transfer coefficient term, [W/K]

Returns

- **results** [dict]
  - **Q** : Heat exchanged in the heat exchanger, [W]
  - **UA** : Combined area-heat transfer coefficient term, [W/K]
  - **Cr** : The heat capacity rate ratio, of the smaller fluid to the larger fluid, [W/K]
  - **Cmin** : The heat capacity rate of the smaller fluid, [W/K]
  - **Cmax** : The heat capacity rate of the larger fluid, [W/K]
  - **effectiveness** : The thermal effectiveness of the heat exchanger, [-]
• NTU : Thermal Number of Transfer Units [-]
• Thi : Inlet temperature of hot fluid, [K]
• Tho : Outlet temperature of hot fluid, [K]
• Tci : Inlet temperature of cold fluid, [K]
• Tco : Outlet temperature of cold fluid, [K]

See also:

effectiveness_from_NTU
NTU_from_effectiveness

Examples

Solve a heat exchanger to determine UA and effectiveness given the configuration, flows, subtype, the cold inlet/outlet temperatures, and the hot stream inlet temperature.

```python
>>> from pprint import pprint
>>> pprint(effectiveness_NTU_method(mh=5.2, mc=1.45, Cph=1860., Cpc=1900,
... subtype='crossflow, mixed Cmax', Tci=15, Tco=85, Thi=130))
{'Cmax': 9672.0,
 'Cmin': 2755.0,
 'Cr': 0.2848428453267163,
 'NTU': 1.1040839095588,
 'Q': 192850.0,
 'Tci': 15,
 'Tco': 85,
 'Thi': 130,
 'Tho': 110.06100082712986,
 'UA': 3041.751170834494,
 'effectiveness': 0.6086956521739131}
```

Solve the same heat exchanger with the UA specified, and known inlet temperatures:

```python
>>> pprint(effectiveness_NTU_method(mh=5.2, mc=1.45, Cph=1860., Cpc=1900,
... subtype='crossflow, mixed Cmax', Tci=15, Thi=130, UA=3041.75))
{'Cmax': 9672.0,
 'Cmin': 2755.0,
 'Cr': 0.2848428453267163,
 'NTU': 1.1040834845735028,
 'Q': 192849.96310220254,
 'Tci': 15,
 'Tco': 84.99998660697007,
 'Thi': 130,
 'Tho': 110.06100464203861,
 'UA': 3041.75,
 'effectiveness': 0.6086955357127832}
```

ht.hx.F_LMTD_Fakheri (Thi, Tho, Tci, Tco, shells=1)
Calculates the log-mean temperature difference correction factor $F_t$ for a shell-and-tube heat exchanger with one or an even number of tube passes, and a given number of shell passes, with the expression given in [1] and also shown in [2].

$$F_t = \frac{S \ln W}{\ln \frac{1+S-W}{1+W-S+SW}}$$
\[ S = \frac{\sqrt{R^2 + 1}}{R - 1} \]
\[ W = \left( \frac{1 - PR}{1 - P} \right)^{1/N} \]
\[ R = \frac{T_{in} - T_{out}}{t_{out} - t_{in}} \]
\[ P = \frac{t_{out} - t_{in}}{T_{in} - t_{in}} \]

If \( R = 1 \) and logarithms cannot be evaluated:

\[ W' = \frac{N - NP}{N - NP + P} \]
\[ F_t = \frac{\sqrt{2^{W' W'}}}{\ln \left( \frac{W' + \frac{1}{W'}}{\frac{1}{W'} - \frac{1}{W'}} \right)} \]

**Parameters**

- **Thi** [float] Inlet temperature of hot fluid, [K]
- **Tho** [float] Outlet temperature of hot fluid, [K]
- **Tci** [float] Inlet temperature of cold fluid, [K]
- **Tco** [float] Outlet temperature of cold fluid, [K]
- **shells** [int, optional] Number of shell-side passes, [-]

**Returns**

- **Ft** [float] Log-mean temperature difference correction factor, [-]

**Notes**

This expression is symmetric - the same result is calculated if the cold side values are swapped with the hot side values. It also does not depend on the units of the temperature given.

**References**

[1], [2]

**Examples**

```python
>>> F_LMTD_Fakheri(Tci=15, Tco=85, Thi=130, Tho=110, shells=1)
0.9438358829645933
```

**ht.hx.temperature_effectiveness_basic**\((RI, NTU1, subtype='crossflow')\)

Returns temperature effectiveness \( PI \) of a heat exchanger with a specified heat capacity ratio, number of transfer units \( NTU1 \), and of type \( subtype \). This function performs the calculations for the basic cases, not actual shell-and-tube exchangers. The supported cases are as follows:

- Counterflow (ex. double-pipe)
- Parallel (ex. double pipe inefficient configuration)
• Crossflow, single pass, fluids unmixed
• Crossflow, single pass, fluid 1 mixed, fluid 2 unmixed
• Crossflow, single pass, fluid 2 mixed, fluid 1 unmixed
• Crossflow, single pass, both fluids mixed

For parallel flow heat exchangers (this configuration is symmetric):

\[ P_1 = \frac{1 - \exp(-NTU_1(1 + R_1))}{1 + R_1} \]

For counterflow heat exchangers (this configuration is symmetric):

\[ P_1 = \frac{1 - \exp(-NTU_1(1 - R_1))}{1 - R_1 \exp(-NTU_1(1 - R_1))} \]

For cross-flow (single-pass) heat exchangers with both fluids unmixed (this configuration is symmetric), there are two solutions available: a frequently cited approximation and an exact solution which uses a numerical integration developed in [4]. The approximate solution is:

\[ P_1 \approx 1 - \exp \left( \frac{NTU_1^{0.32}}{R_1} \left( \exp(-R_1 NTU_1^{0.78}) - 1 \right) \right) \]

The exact solution for crossflow (single pass, fluids unmixed) is:

\[
\epsilon = \frac{1}{R_1} - \frac{\exp(-R_1 \cdot NTU_1)}{2(R_1 \cdot NTU_1)^2} \int_0^{2NTU_1 \sqrt{\pi R_1}} \left( 1 + NTU_1 - \frac{v^2}{4R_1 \cdot NTU_1} \right) \exp \left( -\frac{v^2}{4R_1 \cdot NTU_1} \right) v I_0(v) dv
\]

For cross-flow (single-pass) heat exchangers with fluid 1 mixed, fluid 2 unmixed:

\[ P_1 = 1 - \exp \left( -\frac{K}{R_1} \right) \]

\[ K = 1 - \exp(-R_1 NTU_1) \]

For cross-flow (single-pass) heat exchangers with fluid 2 mixed, fluid 1 unmixed:

\[ P_1 = -\frac{1}{K_1} \exp(-KR_1) \]

\[ K = 1 - \exp(-NTU_1) \]

For cross-flow (single-pass) heat exchangers with both fluids mixed (this configuration is symmetric):

\[ P_1 = \left( \frac{1}{K_1} + \frac{R_1}{K_2} - \frac{1}{NTU_1} \right)^{-1} \]

\[ K_1 = 1 - \exp(-NTU_1) \]

\[ K_2 = 1 - \exp(-R_1 NTU_1) \]

Parameters

- **R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

- **NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]


Returns

- **P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
Notes

The crossflow case is an approximation only. There is an actual solution involving an infinite sum. This was implemented, but found to differ substantially so the approximation is used instead.

References

[1], [2], [3], [4]

Examples

```python
>>> temperature_effectiveness_basic(R1=.1, NTU1=4, subtype='counterflow')
0.9753412729761263
```

ht.hx.temperature_effectiveness_TEMAJ(R1, NTU1, Ntp)

Returns temperature effectiveness $P_1$ of a TEMA J type heat exchanger with a specified heat capacity ratio, number of transfer units $NTU1$, and of number of tube passes $Ntp$. The supported cases are as follows:

- One tube pass (shell fluid mixed)
- Two tube passes (shell fluid mixed, tube pass mixed between passes)
- Four tube passes (shell fluid mixed, tube pass mixed between passes)

For 1-1 TEMA J shell and tube exchangers, shell and tube fluids mixed:

$$P_1 = \frac{1}{R_1} \left[ 1 - \frac{(2 - R_1)(2E + R_1B)}{(2 + R_1)(2E - R_1/B)} \right]$$

For 1-2 TEMA J, shell and tube fluids mixed. There are two possible arrangements for the flow and the number of tube passes, but the equation is the same in both:

$$P_1 = \left[ 1 + R_1 \left( \frac{1}{2} + \lambda B - 2\lambda CD \right) \right]^{-1}$$

$$B = \frac{(A^\lambda + 1)}{A^\lambda - 1}$$

$$C = \frac{A^{(1+\lambda)/2}}{\lambda - 1 + (1 + \lambda)A^\lambda}$$

$$D = 1 + \frac{\lambda A^{(\lambda-1)/2}}{A^\lambda - 1}$$

$$A = \exp(NTU_1)$$

$$\lambda = (1 + R_2^2/4)^{0.5}$$

For 1-4 TEMA J, shell and tube exchanger with both sides mixed:

$$P_1 = \left[ 1 + \frac{R_1}{4} \left( \frac{1 + 3E}{1 + E} \right) + \lambda B - 2\lambda CD \right]^{-1}$$

$$B = \frac{A^\lambda + 1}{A^\lambda - 1}$$

$$C = \frac{A^{(1+\lambda)/2}}{\lambda - 1 + (1 + \lambda)A^\lambda}$$
\[
D = 1 + \frac{\lambda A^{(\lambda - 1)/2}}{A^\lambda - 1} \\
A = \exp(NTU_1) \\
E = \exp(R_1 NTU_1/2) \\
\lambda = (1 + R_1^2/16)^{0.5}
\]

Parameters

\textbf{R1} [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

\textbf{NTU1} [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

\textbf{Ntp} [int] Number of tube passes, 1, 2, or 4, [-]

Returns

\textbf{P1} [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

Notes

For numbers of tube passes that are not 1, 2, or 4, an exception is raised. The convention for the formulas in [1] and [3] are with the shell side as side 1, and the tube side as side 2. [2] has formulas with the opposite convention.

References

[1], [2], [3]

Examples

```python
temperature_effectiveness_TEMA_J(R1=1/3., NTU1=1., Ntp=1)
0.5699085193651295
```

\texttt{ht.hx.temperature_effectiveness\_TEMA\_H} \texttt{(R1, NTU1, Ntp, optimal=True)}

Returns temperature effectiveness \textit{P1} of a TEMA H type heat exchanger with a specified heat capacity ratio, number of transfer units \textit{NTU1}, and of number of tube passes \textit{Ntp}. For the two tube pass case, there are two possible orientations, one inefficient and one efficient controlled by the \textit{optimal} option. The supported cases are as follows:

- One tube pass (tube fluid split into two streams individually mixed, shell fluid mixed)
- Two tube passes (shell fluid mixed, tube pass mixed between passes)
- Two tube passes (shell fluid mixed, tube pass mixed between passes, inlet tube side next to inlet shell-side)

1-1 TEMA H, tube fluid split into two streams individually mixed, shell fluid mixed:

\[
P_1 = E[1 + (1 - BR_1/2)(1 - AR_1/2 + ABR_1)] - AB(1 - BR_1/2)
A = \frac{1}{1 + R_1/2} \{1 - \exp[-NTU_1(1 + R_1/2)/2]\}
\]
\[ B = \frac{1 - D}{1 - R_1 D/2} \]
\[ D = \exp[-NTU_1(1 - R_1/2)/2] \]
\[ E = (A + B - ABR_1/2)/2 \]

1-2 TEMA H, shell and tube fluids mixed in each pass at the cross section:

\[ P_1 = \frac{1}{R_1} \left[ 1 - \frac{(1 - D)^4}{B - 4G/R_1} \right] \]
\[ B = (1 + H)(1 + E)^2 \]
\[ G = (1 - D)^2(D^2 + E^2) + D^2(1 + E)^2 \]
\[ H = [1 - \exp(-2\beta)]/(4/R_1 - 1) \]
\[ E = [1 - \exp(-\beta)]/(4/R_1 - 1) \]
\[ D = [1 - \exp(-\alpha)]/(4/R_1 + 1) \]
\[ \alpha = NTU_1(4 + R_1)/8 \]
\[ \beta = NTU_1(4 - R_1)/8 \]

1-2 TEMA H, shell and tube fluids mixed in each pass at the cross section but with the inlet tube stream coming in next to the shell fluid inlet in an inefficient way (this is only shown in [2], and the stream 1/2 convention in it is different but converted here; P1 is still returned):

\[ P_2 = \left[ 1 - \frac{B + 4GR_2}{(1 - D)^4} \right] \]
\[ B = (1 + H)(1 + E)^2 \]
\[ G = (1 - D)^2(D^2 + E^2) + D^2(1 + E)^2 \]
\[ D = \frac{1 - \exp(-\alpha)}{1 - 4R_2} \]
\[ E = \frac{\exp(-\beta) - 1}{4R_2 + 1} \]
\[ H = \frac{\exp(-2\beta) - 1}{4R_2 + 1} \]
\[ \alpha = \frac{NTU_2}{8}(4R_2 - 1) \]
\[ \beta = \frac{NTU_2}{8}(4R_2 + 1) \]

Parameters

- **R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- **NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- **Ntp** [int] Number of tube passes, 1, or 2, [-]
- **optimal** [bool, optional] Whether or not the arrangement is configured to give more of a countercurrent and efficient (True) case or an inefficient parallel case, [-]

Returns

- **P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
Notes

For numbers of tube passes greater than 1 or 2, an exception is raised. The convention for the formulas in [1] and [3] are with the shell side as side 1, and the tube side as side 2. [2] has formulas with the opposite convention.

References

[1], [2], [3]

Examples

```python
>>> temperature_effectiveness_TEMA_H(R1=1/3., NTU1=1., Ntp=1)
0.5730728284905833
```

`.ht.hx.temperature_effectiveness_TEMA_G(R1, NTU1, Ntp, optimal=True)`

Returns temperature effectiveness \( P_1 \) of a TEMA G type heat exchanger with a specified heat capacity ratio, number of transfer units \( NTU1 \), and of number of tube passes \( Ntp \). For the two tube pass case, there are two possible orientations, one inefficient and one efficient controlled by the `optimal` option. The supported cases are as follows:

- One tube pass (tube fluid split into two streams individually mixed, shell fluid mixed)
- Two tube passes (shell and tube exchanger with shell and tube fluids mixed in each pass at the cross section), counterflow arrangement
- Two tube passes (shell and tube exchanger with shell and tube fluids mixed in each pass at the cross section), parallelflow arrangement

1-1 TEMA G, tube fluid split into two streams individually mixed, shell fluid mixed (this configuration is symmetric):

\[
\begin{align*}
P_1 &= A + B - AB(1 + R_1) + R_1 AB^2 \\
A &= \frac{1}{1 + R_1} \{1 - \exp(-NTU_1(1 + R_1)/2)\} \\
B &= \frac{1 - D}{1 - R_1 D} \\
D &= \exp[-NTU_1(1 - R_1)/2]
\end{align*}
\]

1-2 TEMA G, shell and tube exchanger with shell and tube fluids mixed in each pass at the cross section:

\[
\begin{align*}
P_1 &= (B - \alpha^2)/(A + 2 + R_1 B) \\
A &= -2R_1(1 - \alpha)^2/(2 + R_1) \\
B &= [4 - \beta(2 + R_1)]/(2 - R_1) \\
\alpha &= \exp[-NTU_1(2 + R_1)/4] \\
\beta &= \exp[-NTU_1(2 - R_1)/2]
\end{align*}
\]

1-2 TEMA G, shell and tube exchanger in overall parallelflow arrangement with shell and tube fluids mixed in each pass at the cross section (this is only shown in [2], and the stream convention in it is different but converted here; \( P_1 \) is still returned):

\[
P_2 = \frac{(B - \alpha^2)}{R_2(A - \alpha^2/R_2 + 2)}
\]
\[
A = \frac{(1 - \alpha)^2}{(R_2 - 0.5)} \\
B = \frac{4R_2 - \beta(2R_2 - 1)}{2R_2 + 1} \\
\alpha = \exp\left(-\frac{NTU_2(2R_2 - 1)}{4}\right) \\
\beta = \exp\left(-\frac{NTU_2(2R_2 + 1)}{2}\right)
\]

Parameters

- **R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- **NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- **Ntp** [int] Number of tube passes, 1 or 2, [-]
- **optimal** [bool, optional] Whether or not the arrangement is configured to give more of a countercurrent and efficient (True) case or an inefficient parallel case (only applies for two passes), [-]

Returns

- **P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

Notes

For numbers of tube passes greater than 1 or 2, an exception is raised. The convention for the formulas in [1] and [3] are with the shell side as side 1, and the tube side as side 2. [2] has formulas with the opposite convention.

References

[1], [2], [3]

Examples

```python
>>> temperature_effectiveness_TEMA_G(R1=1/3., NTU1=1., Ntp=1)
0.5730149350867675
```

`ht.hx.temperature_effectiveness_TEMA_E(R1, NTU1, Ntp=1, optimal=True)`

Returns temperature effectiveness \( P1 \) of a TEMA E type heat exchanger with a specified heat capacity ratio, number of transfer units \( NTU1 \), number of tube passes \( Ntp \), and whether or not it is arranged in a more countercurrent (optimal configuration) way or a more parallel (optimal=True) case. The supported cases are as follows:

- 1-1 TEMA E, shell fluid mixed
- 1-2 TEMA E, shell fluid mixed (this configuration is symmetric)
- 1-2 TEMA E, shell fluid split into two steams individually mixed
- 1-3 TEMA E, shell and tube fluids mixed, one parallel pass and two counterclockwise passes (efficient)
- 1-3 TEMA E, shell and tube fluids mixed, two parallel passes and one counterclockwise pass (inefficient)
• 1-N TEMA E, shall and tube fluids mixed, efficient counterflow orientation, N an even number

1-1 TEMA E, shell fluid mixed:

\[ P_1 = \frac{1 - \exp[-NTU_1(1-R_1)]}{1 - R_1 \exp[-NTU_1(1-R_1)]} \]

1-2 TEMA E, shell fluid mixed (this configuration is symmetric):

\[ P_1 = \frac{2}{1 + R_1 + E \coth(E \cdot NTU_1/2)} \]

\[ E = [1 + R_1^2]^{1/2} \]

1-2 TEMA E, shell fluid split into two steams individually mixed:

\[ P_1 = \frac{1}{R_1} \left[ 1 - \frac{(2-R_1)(2E + R_1B)}{(2 + R_1)(2E - R_1/B)} \right] \]

\[ E = \exp(NTU_1) \]

\[ B = \exp(-NTU_1R_1/2) \]

1-3 TEMA E, shell and tube fluids mixed, one parallel pass and two counterflow passes (efficient):

\[ P_1 = \frac{1}{R_1} \left[ 1 - \frac{C}{AC + B^2} \right] \]

\[ A = X_1(R_1 + \lambda_1)(R_1 - \lambda_2)/(2\lambda_1) - X_3\delta - X_2(R_1 + \lambda_2)(R_1 - \lambda_1)/(2\lambda_2) + 1/(1 - R_1) \]

\[ B = X_1(R_1 - \lambda_2) - X_2(R_1 - \lambda_1) + X_3\delta \]

\[ C = X_2(3R_1 + \lambda_1) - X_1(3R_1 + \lambda_2) + X_3\delta \]

\[ X_i = \exp(\lambda_iNTU_1/3)/(2\delta), \ i = 1, 2, 3 \]

\[ \delta = \lambda_1 - \lambda_2 \]

\[ \lambda_1 = -\frac{3}{2} + \left[ \frac{9}{4} + R_1(R_1 - 1) \right]^{1/2} \]

\[ \lambda_2 = -\frac{3}{2} - \left[ \frac{9}{4} + R_1(R_1 - 1) \right]^{1/2} \]

\[ \lambda_3 = R_1 \]

1-3 TEMA E, shell and tube fluids mixed, two parallel passes and one counteflow pass (inefficient):

\[ P_2 = \left[ 1 - \frac{C}{(AC + B^2)} \right] \]

\[ A = \chi_1(1 + R_2\lambda_1)(1 - R_2\lambda_2)/(2R_2^2\lambda_1) - E - \chi_2(1 + R_2\lambda_2)(1 - R_2\lambda_1)/(2R_2^2\lambda_2) + R/(R - 1) \]

\[ B = \chi_1(1 - R_2\lambda_2)/R_2 - \chi_2(1 - R_2\lambda_1)/R_2 + E \]

\[ C = -\chi_1(3 + R_2\lambda_2)/R_2 + \chi_2(3 + R_2\lambda_1)/R_2 + E \]

\[ E = 0.5 \exp(NTU_2/3) \]

\[ \lambda_1 = (-3 + \delta)/2 \]

\[ \lambda_2 = (-3 - \delta)/2 \]
\[
\delta = \left[9R_2^2 + 4(1 - R_2)\right]^{0.5} \\
\chi_1 = \exp(\lambda_1 R_2 NTU_2/3) / 2\delta \\
\chi_2 = \exp(\lambda_2 R_2 NTU_2/3) / 2\delta
\]

1-N TEMA E, shell and tube fluids mixed, efficient counterflow orientation, N an even number:

\[
P_2 = \frac{2}{A + B + C}
\]

\[
A = 1 + R_2 + \coth(NTU_2/2)
\]

\[
B = -\frac{1}{N_1} \coth \left(\frac{NTU_2}{2N_1}\right)
\]

\[
C = \frac{1}{N_1} \sqrt{1 + N_1^2 R_2^2} \coth \left(\frac{NTU_2}{2N_1} \sqrt{1 + N_1^2 R_2^2}\right)
\]

\[N_1 = \frac{N_{tp}}{2}\]

Parameters

- **R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- **NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- **Ntp** [int] Number of tube passes, 1, 2, 3, 4, or an even number[-]
- **optimal** [bool, optional] Whether or not the arrangement is configured to give more of a countercurrent and efficient (True) case or an inefficient parallel case, [-]

Returns

- **P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

Notes

For odd numbers of tube passes greater than 3, an exception is raised. [2] actually has a formula for 5 tube passes, but it is an entire page long. The convention for the formulas in [1] and [3] are with the shell side as side 1, and the tube side as side 2. [2] has formulas with the opposite convention.

References

[1], [2], [3]

Examples

```python
>>> temperature_effectiveness_TEMA_E(R1=1/3., NTU1=1., Ntp=1)
0.5870500654031314
```
ht.hx.temperature_effectiveness_plate(R1, NTU1, Np1, Np2, counterflow=True, passes_counterflow=True, reverse=False)

Returns the temperature effectiveness $P_1$ of side 1 of a plate heat exchanger with a specified side 1 heat capacity ratio $R_1$, side 1 number of transfer units $NTU1$, number of passes on sides 1 and 2 (respectively $Np1$ and $Np2$).

For all cases, the function also takes as arguments whether the exchanger is setup in an overall counter or parallel orientation counterflow, and whether or not individual stream passes are themselves counterflow or parallel.

The 20 supported cases are as follows. (the first number of sides listed refers to side 1, and the second number refers to side 2):

- 1 pass/1 pass parallel flow
- 1 pass/1 pass counterflow
- 1 pass/2 pass
- 1 pass/3 pass or 3 pass/1 pass (with the two end passes in parallel)
- 1 pass/3 pass or 3 pass/1 pass (with the two end passes in counterflow)
- 1 pass/4 pass
- 2 pass/2 pass, overall parallel flow, individual passes in parallel
- 2 pass/2 pass, overall parallel flow, individual passes counterflow
- 2 pass/2 pass, overall counterflow, individual passes parallel flow
- 2 pass/2 pass, overall counterflow, individual passes counterflow
- 2 pass/3 pass or 3 pass/2 pass, overall parallel flow
- 2 pass/3 pass or 3 pass/2 pass, overall counterflow
- 2 pass/4 pass or 4 pass/2 pass, overall parallel flow
- 2 pass/4 pass or 4 pass/2 pass, overall counterflow flow

For all plate heat exchangers, there are two common formulas used by most of the expressions.

$$P_p(x, y) = \frac{1 - \exp[-x(1 + y)]}{1 + y}$$

$$P_c(x, y) = \frac{1 - y \exp[-x(1 - y)]}{1 - y \exp[-x(1 - y)]}$$

The main formulas used are as follows. Note that for some cases such as 4 pass/2 pass, the formula is not shown because it is that of 2 pass/4 pass, but with $R1$, $NTU1$, and $P1$ conversions.

For 1 pass/1 pass parallel flow (streams symmetric):

$$P_1 = P_p(NTU_1, R_1)$$

For 1 pass/1 pass counterflow (streams symmetric):

$$P_1 = P_c(NTU_1, R_1)$$

For 1 pass/2 pass (any of the four possible configurations):

$$P_1 = 0.5(A + B - 0.5ABR_1)$$

$$A = P_p(NTU_1, 0.5R_1)$$

$$B = P_c(NTU_1, 0.5R_1)$$
For 1 pass/3 pass (with the two end passes in parallel):

\[
P_1 = \frac{1}{3} \left[ B + A \left( 1 - \frac{R_1 B}{3} \right) \left( 2 - \frac{R_1 A}{3} \right) \right]
\]

\[
A = P_p \left( NTU_1, \frac{R_1}{3} \right)
\]

\[
B = P_c \left( NTU_1, \frac{R_1}{3} \right)
\]

For 1 pass/3 pass (with the two end passes in counterflow):

\[
P_1 = \frac{1}{3} \left[ A + B \left( 1 - \frac{R_1 A}{3} \right) \left( 2 - \frac{R_1 B}{3} \right) \right]
\]

\[
A = P_p \left( NTU_1, \frac{R_1}{3} \right)
\]

\[
B = P_c \left( NTU_1, \frac{R_1}{3} \right)
\]

For 1 pass/4 pass (any of the four possible configurations):

\[
P_1 = \frac{1 - Q}{R_1}
\]

\[
Q = \left( 1 - \frac{AR_1}{4} \right)^2 \left( 1 - \frac{BR_1}{4} \right)^2
\]

\[
A = P_p \left( NTU_1, \frac{R_1}{4} \right)
\]

\[
B = P_c \left( NTU_1, \frac{R_1}{4} \right)
\]

For 2 pass/2 pass, overall parallelflow, individual passes in parallel (stream symmetric):

\[
P_1 = P_p(NTU_1, R_1)
\]

For 2 pass/2 pass, overall parallelflow, individual passes counterflow (stream symmetric):

\[
P_1 = B[2 - B(1 + R_1)]
\]

\[
B = P_c \left( \frac{NTU_1}{2}, R_1 \right)
\]

For 2 pass/2 pass, overall counterflow, individual passes parallelflow (stream symmetric):

\[
P_1 = \frac{2A - A^2(1 + R_1)}{1 - R_1 A^2}
\]

\[
A = P_p \left( NTU_1, \frac{R_1}{2} \right)
\]

For 2 pass/2 pass, overall counterflow and individual passes counterflow (stream symmetric):

\[
P_1 = P_c(NTU_1, R_1)
\]

For 2 pass/3 pass, overall parallelflow:

\[
P_1 = A + B - \left( \frac{2}{9} + \frac{D}{3} \right) \left( A^2 + B^2 \right) - \left( \frac{5}{9} + \frac{4D}{3} \right) AB + \frac{D(1 + D)AB(A + B)}{3} - \frac{D^2 A^2 B^2}{9}
\]
\[ A = P_p \left( \frac{NTU_1}{2}, D \right) \]
\[ B = P_c \left( \frac{NTU_1}{2}, D \right) \]
\[ D = \frac{2R_1}{3} \]

For 2 pass/3 pass, overall counterflow:
\[ P_1 = \frac{A + 0.5B + 0.5C + D}{R_1} \]
\[ A = \frac{2R_1EF^2 - 2EF + F - F^2}{2R_1E^2F^2 - E^2 - F^2 - 2EF + E + F} \]
\[ B = \frac{A(E - 1)}{F} \]
\[ C = \frac{1 - A}{E} \]
\[ D = R_1E^2C - R_1E + R_1 - \frac{C}{2} \]
\[ E = \frac{3}{2R_1G} \]
\[ F = \frac{3}{2R_1H} \]
\[ G = P_c \left( \frac{NTU_1}{2}, \frac{2R_1}{3} \right) \]
\[ H = P_p \left( \frac{NTU_1}{2}, \frac{2R_1}{3} \right) \]

For 2 pass/4 pass, overall parallel flow:
\[ P_1 = 2D - (1 + R_1)D^2 \]
\[ D = \frac{A + B - 0.5ABR_1}{2} \]
\[ A = P_p \left( \frac{NTU_1}{2}, \frac{R_1}{2} \right) \]
\[ B = P_c \left( \frac{NTU_1}{2}, \frac{R_1}{2} \right) \]

For 2 pass/4 pass, overall counterflow flow:
\[ P_1 = \frac{2D - (1 + R_1)D^2}{1 - D^2R_1} \]
\[ D = \frac{A + B - 0.5ABR_1}{2} \]
\[ A = P_p \left( \frac{NTU_1}{2}, \frac{R_1}{2} \right) \]
\[ B = P_c \left( \frac{NTU_1}{2}, \frac{R_1}{2} \right) \]

Parameters
**R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

**NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

**Np1** [int] Number of passes on side 1 [-]

**Np2** [int] Number of passes on side 2 [-]

**counterflow** [bool] Whether or not the overall flow through the heat exchanger is in counterflow or parallel flow, [-]

**passes_counterflow** [bool] In addition to the overall flow direction, in some cases individual passes may be in counter or parallel flow; this controls that [-]

**reverse** [bool] Used internally only to allow cases like the 1-4 formula to work for the 4-1 flow case, without having to duplicate the code [-]

**Returns**

**P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

**Notes**

For diagrams of these heat exchangers, see [3]. In all cases, each pass is assumed to be made up of an infinite number of plates. The fluid velocities must be uniform across the plate channels, and the flow must be uniformly distributed between the channels. The heat transfer coefficient is also assumed constant.

The defaults of counterflow=True and passes_counterflow=True will always result in the most efficient heat exchanger option, normally what is desired.

If a number of passes which is not supported is provided, an exception is raised.

**References**

[1], [2], [3], [4]

**Examples**

Three passes on side 1; one pass on side 2; two end passes in counterflow orientation.

```python
>>> temperature_effectiveness_plate(R1=1/3., NTU1=1., Np1=3, Np2=1)
0.5743514352720835
```

If the same heat exchanger (in terms of NTU1 and R1) were operating with sides 1 and 2 switched, a slightly less efficient design results.

```python
>>> temperature_effectiveness_plate(R1=1/3., NTU1=1., Np1=1, Np2=3)
0.5718726757657066
```

ht.hx.temperature_effectiveness_air_cooler(R1, NTU1, rows, passes, coerce=True)

Returns temperature effectiveness *P1* of an air cooler with a specified heat capacity ratio, number of transfer units *NTU1*, number of rows *rows*, and number of passes *passes*. The supported cases are as follows:

- N rows 1 pass
- N row N pass (up to N = 5)
• 4 rows 2 passes

For N rows 1 passes ([12], shown in [1] and [3]):

\[
P = \frac{1}{R} \left\{ 1 - \frac{N \exp(NKR)}{1 + \sum_{i=1}^{N-1} \sum_{j=0}^{i} (\frac{i}{j})K^j \exp(-(i-j)NTU/N) \sum_{k=0}^{j} (\frac{NKR}{k!})} \right\}^{-1}
\]

For 2 rows 2 passes (cited as from [4] in [1]):

\[
P_1 = \frac{1}{R} \left( 1 - \frac{1}{\xi} \right)
\]

\[
\xi = \frac{K}{2} + \left( 1 - \frac{K}{2} \right) \exp(2KR)
\]

\[
K = 1 - \exp\left( -\frac{NTU}{2} \right)
\]

For 3 rows / 3 passes (cited as from [4] in [1]):

\[
\xi = K \left[ 1 - \frac{K}{4} - RK \left( 1 - \frac{K}{2} \right) \right] \exp(KR) + \exp(3KR) \left( 1 - \frac{K}{2} \right)^2
\]

\[
K = 1 - \exp\left( -\frac{NTU}{3} \right)
\]

For 4 rows / 4 passes (cited as from [4] in [1]):

\[
\xi = \frac{K}{2} \left( 1 - \frac{K}{2} + \frac{K^2}{4} \right) + K \left[ 1 - \frac{R}{8} K \left( 1 - \frac{K}{2} \right) \exp(2KR) \right] + \exp(4KR) \left( 1 - \frac{K}{2} \right)^3
\]

\[
K = 1 - \exp\left( -\frac{NTU}{4} \right)
\]

For 5 rows / 5 passes (cited as from [4] in [1]):

\[
\xi = \left\{ K \left( 1 - \frac{3}{4} K + \frac{K^2}{2} - \frac{K^3}{8} \right) - RK^2 \left[ 1 - K + \frac{3}{4} K^2 - \frac{1}{4} K^3 - \frac{R}{2} K^2 \left( 1 - \frac{K}{2} \right)^2 \right] \right\} \exp(KR) + \left[ K \left( 1 - \frac{3}{4} K + \frac{1}{16} K^3 \right) \right]
\]

For 4 rows / 2 passes (cited as from [4] in [1]):

\[
P_1 = \frac{1}{R} \left( 1 - \frac{1}{\xi} \right)
\]

\[
\xi = \left\{ \frac{R}{2} K^3 [4 - K + 2RK^2] + \exp(4KR) + K \left[ 1 - \frac{K}{2} + \frac{K^2}{8} \right] \left( 1 - \exp(4KR) \right) \right\} \frac{1}{(1 + RK^2)^2}
\]

\[
K = 1 - \exp\left( -\frac{NTU}{4} \right)
\]

Parameters

**R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (process fluid side) [-]

**NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (process fluid side) [-]

**rows** [int] Number of rows of tubes in the air cooler [-]

**passes** [int] Number of passes the process fluid undergoes [-]
coerce [bool] If True, the number of passes or rows, if otherwise unsupported, will be replaced with a similar number to allow the calculation to proceed, [-]

Returns

P1 [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (process fluid side) [-]

Notes

For the 1-pass case, the exact formula used can take a while to compute for large numbers of tube rows; 100 us for 20 rows, 1 ms for 50 rows. Floating point rounding behavior can also be an issue for large numbers of tube passes, leading to thermal effectivenesses larger than one being returned:

```
>>> temperature_effectiveness_air_cooler(1e-10, 100, rows=150, passes=1.0)
1.000026728092962
```

Furthermore, as a factorial of the number of tube counts is used, there comes a point where standard floats are not able to hold the intermediate calculations values and an error will occur:

```
>>> temperature_effectiveness_air_cooler(.5, 1.1, rows=200, passes=1.0)
Traceback (most recent call last):
...
OverflowError: int too large to convert to float
```

References

[1], [2], [3], [4]

Examples

```
>>> temperature_effectiveness_air_cooler(.5, 2, rows=2, passes=2)
0.7523072855817072
```

ht.hx.P_NTU_method(m1, m2, Cp1, Cp2, UA=None, T1i=None, T1o=None, T2i=None, T2o=None, subtype='crossflow', Ntp=1, optimal=True)

Wrapper for the various P-NTU method function calls, which can solve a heat exchanger. The heat capacities and mass flows of each stream and the type of the heat exchanger are always required. As additional inputs, one combination of the following inputs is required:

- Three of the four inlet and outlet stream temperatures.
- Temperatures for the side 1 outlet and side 2 inlet and UA
- Temperatures for the side 1 outlet side 2 outlet and UA
- Temperatures for the side 1 inlet and side 2 inlet and UA
- Temperatures for the side 1 inlet and side 2 outlet and UA

Computes the total heat exchanged as well as both temperatures of both streams.

Parameters

m1 [float] Mass flow rate of stream 1 (shell side = 1, tube side = 2), [kg/s]

m2 [float] Mass flow rate of stream 2 (shell side = 1, tube side = 2), [kg/s]
Heat Transfer Documentation, Release 0.1.56

Cp1 [float] Averaged heat capacity of stream 1 (shell side), [J/kg/K]

Cp2 [float] Averaged heat capacity of stream 2 (tube side), [J/kg/K]

UA [float, optional] Combined Area-heat transfer coefficient term, [W/K]

T1i [float, optional] Inlet temperature of stream 1 (shell side), [K]

T1o [float, optional] Outlet temperature of stream 1 (shell side), [K]

T2i [float, optional] Inlet temperature of stream 2 (tube side), [K]

T2o [float, optional] Outlet temperature of stream 2 (tube-side), [K]


Ntp [int, optional] For real heat exchangers (types ‘E’, ‘G’, ‘H’, and ‘J’), the number of tube passes needs to be specified as well. Not all types support any number of tube passes.

optimal [bool, optional] For real heat exchangers (types ‘E’, ‘G’, ‘H’, and ‘J’), there is often a more countercurrent (optimal) way to arrange the tube passes and a more parallel (optimal=False) way to arrange them. This controls that.

Returns

results [dict]

• Q : Heat exchanged in the heat exchanger, [W]

• UA : Combined area-heat transfer coefficient term, [W/K]

• T1i : Inlet temperature of stream 1, [K]

• T1o : Outlet temperature of stream 1, [K]

• T2i : Inlet temperature of stream 2, [K]

• T2o : Outlet temperature of stream 2, [K]

• P1 : Thermal effectiveness with respect to stream 1, [-]

• P2 : Thermal effectiveness with respect to stream 2, [-]

• R1 : Heat capacity ratio with respect to stream 1, [-]

• R2 : Heat capacity ratio with respect to stream 2, [-]

• C1 : The heat capacity rate of fluid 1, [W/K]

• C2 : The heat capacity rate of fluid 2, [W/K]

• NTU1 : Thermal Number of Transfer Units with respect to stream 1 [-]

• NTU2 : Thermal Number of Transfer Units with respect to stream 2 [-]

See also:

temperature_effectiveness_basic
temperature_effectiveness_plate
temperature_effectiveness_TEMA_E
temperature_effectiveness_TEMA_G
temperature_effectiveness_TEMA_H

2.18. Heat exchanger sizing and rating (ht.hx) 199
The main equations used in this method are as follows. For the individual expressions used to calculate $P_1$, see the See Also section.

$$Q = P_1 C_1 \Delta T_{max} = P_2 C_2 \Delta T_{max}$$

$$\Delta T_{max} = T_{h,i} - T_{c,i} = |T_{2,i} - T_{1,i}|$$

$$R_1 = \frac{C_1}{C_2} = \frac{T_{2,i} - T_{2,o}}{T_{1,o} - T_{1,i}}$$

$$R_2 = \frac{C_2}{C_1} = \frac{T_{1,o} - T_{1,i}}{T_{2,i} - T_{2,o}}$$

$$R_1 = \frac{1}{R_2}$$

$$NTU_1 = \frac{UA}{C_1}$$

$$NTU_2 = \frac{UA}{C_2}$$

$$NTU_1 = NTU_2 R_2$$

$$NTU_2 = NTU_1 R_1$$

$$P_1 = \frac{T_{1,o} - T_{1,i}}{T_{2,i} - T_{1,i}}$$

$$P_2 = \frac{T_{2,i} - T_{2,o}}{T_{2,i} - T_{1,i}}$$

$$P_1 = P_2 R_2$$

$$P_2 = P_1 R_1$$

$$C_1 = m_1 C p_1$$

$$C_2 = m_2 C p_2$$

Once $P_1$ has been calculated, there are six different cases for calculating the other stream temperatures depending on the two temperatures provided. They were derived with SymPy.

Two known inlet temperatures:

$$T_{1,o} = -P_1 T_{1,i} + P_1 T_{2,i} + T_{1,i}$$
\[ T_{2,o} = P_1 R_1 T_{1,i} - P_1 R_1 T_{2,i} + T_{2,i} \]

Two known outlet temperatures:

\[
T_{1,i} = \frac{P_1 R_1 T_{1,o} + P_2 T_{2,o} - T_{1,o}}{P_1 R_1 + P_2 - 1} \\
T_{2,i} = \frac{P_1 R_1 T_{1,o} + P_2 T_{2,o} - T_{2,o}}{P_1 R_1 + P_2 - 1}
\]

Inlet 1 known, outlet 2 known:

\[
T_{1,o} = \frac{1}{P_1 R_1 - 1} (P_1 R_1 T_{1,i} + P_1 T_{1,i} - P_2 T_{2,o} - T_{1,i})
\]

\[
T_{2,i} = \frac{P_1 R_1 T_{1,i} - T_{2,o}}{P_1 R_1 - 1}
\]

Outlet 1 known, inlet 2 known:

\[
T_{1,i} = \frac{P_1 T_{2,i} - T_{1,o}}{P_1 - 1}
\]

\[
T_{2,o} = \frac{1}{P_1 - 1} \left( R_1 (P_1 T_{2,i} - T_{1,o}) - (P_1 - 1) (R_1 T_{1,o} - T_{2,i}) \right)
\]

Input and output of 2 known:

\[
T_{1,i} = \frac{1}{P_1 R_1} (P_1 R_1 T_{2,i} - T_{2,i} + T_{2,o})
\]

\[
T_{1,o} = \frac{1}{P_1 R_1} (P_1 R_1 T_{2,i} + (P_1 - 1) (T_{2,i} - T_{2,o}))
\]

Input and output of 1 known:

\[
T_{2,i} = \frac{1}{P_1} (P_1 T_{1,i} - T_{1,i} + T_{1,o})
\]

\[
T_{2,o} = \frac{1}{P_1} (P_1 R_1 (T_{1,i} - T_{1,o}) + P_1 T_{1,i} - T_{1,i} + T_{1,o})
\]

References

[1], [2], [3]

Examples

Solve a heat exchanger with the UA specified, and known inlet temperatures:

```python
>>> from pprint import pprint

>>> pprint(P_NTU_method(m1=5.2, m2=1.45, Cp1=1860., Cp2=1900,
... subtype='E', Ntp=4, T2i=15, T1i=130, UA=3041.75))
{'C1': 9672.0, 'C2': 2755.0, 'NTU1': 0.3144902812241522, 'NTU2': 1.1040834845735028, 'P1': 0.1730811614360235, 'P2': 0.6076373841775751, 'Q': 192514.71424206023,}
```

(continues on next page)
Solve the same heat exchanger as if \( T_{1i}, T_{2i}, \) and \( T_{2o} \) were known but \( UA \) was not:

```python
>>> pprint(P_NTU_method(m1=5.2, m2=1.45, Cp1=1860., Cp2=1900, subtype='E',
... Ntp=4, T1i=130, T2i=15, T2o=84.87829918042112))
{'C1': 9672.0,
 'C2': 2755.0,
 'NTU1': 0.3144902812236521,
 'NTU2': 1.1040834845717469,
 'P1': 0.17308116143602348,
 'P2': 0.607637384177575,
 'Q': 192514.7142420602,
 'R1': 3.5107078039927404,
 'R2': 0.2848428453267163,
 'T1i': 130,
 'T1o': 110.09566643485729,
 'T2i': 15,
 'T2o': 84.87829918042112,
 'UA': 3041.75}
```

Solve a 2 pass/2 pass plate heat exchanger with overall parallel flow and its individual passes operating in parallel and known outlet temperatures. Note the overall parallel part is triggered with `optimal=False`, and the individual pass parallel is triggered by appending ‘p’ to the subtype. The subpass counterflow can be specified by appending ‘c’ instead to the subtype, but this is never necessary as it is the default.

```python
>>> pprint(P_NTU_method(m1=5.2, m2=1.45, Cp1=1860., Cp2=1900., UA=300,
... T1o=126.7, T2o=26.7, subtype='2/2p', optimal=False))
{'C1': 9672.0,
 'C2': 2755.0,
 'NTU1': 0.031017369727047148,
 'NTU2': 0.1088929219600726,
 'P1': 0.028945295974795074,
 'P2': 0.10161847646759273,
 'Q': 32200.050307849266,
 'R1': 3.5107078039927404,
 'R2': 0.284828453267163,
 'T1i': 130.02920288542694,
 'T1o': 126.7,
 'T2i': 15.012141449056527,
 'T2o': 26.7,
 'UA': 300}
```

\[ h_t.hx.NTU_from_P_basic(P1, R1, subtype='\text{crossflow}') \]

Returns the number of transfer units of a basic heat exchanger type with a specified (for side 1) thermal effectiveness \( P1 \), and heat capacity ratio \( R1 \). The supported cases are as follows:

- Counterflow (ex. double-pipe) [analytical]
- Parallel (ex. double pipe inefficient configuration) [analytical]
- Crossflow, single pass, fluids unmixed [numerical]
• Crossflow, single pass, fluid 1 mixed, fluid 2 unmixed [analytical]
• Crossflow, single pass, fluid 2 mixed, fluid 1 unmixed [analytical]
• Crossflow, single pass, both fluids mixed [numerical]

The analytical solutions, for those cases they are available, are as follows:

**Counterflow:**

\[ NTU_1 = -\frac{1}{R_1} \log \left( \frac{P_1 R_1 - 1}{P_1 - 1} \right) \]

**Parallel:**

\[ NTU_1 = \frac{1}{R_1 + 1} \log \left( -\frac{1}{P_1 (R_1 + 1) - 1} \right) \]

**Crossflow, single pass, fluid 1 mixed, fluid 2 unmixed:**

\[ NTU_1 = -\frac{1}{R_1} \log \left( R_1 \log \left( -\left( P_1 - 1 \right) e^{\frac{1}{R_1}} \right) \right) \]

**Crossflow, single pass, fluid 2 mixed, fluid 1 unmixed**

\[ NTU_1 = -\log \left( \frac{1}{R_1} \log \left( -\left( P_1 R_1 - 1 \right) e^{R_1} \right) \right) \]

**Parameters**

- **P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
- **R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

**Returns**

- **NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

**Notes**

Although this function allows the thermal effectiveness desired to be specified, it does not mean such a high value can be obtained. An exception is raised when this occurs, although not always a helpful one.

```python
>>> NTU_from_P_basic(P1=.99, R1=.1, subtype='parallel')
ValueError: math domain error
```

For the ‘crossflow approximate’ solution the function is monotonic, and a bounded solver is used within the range of NTU1 from 1E-11 to 1E5.

For the full correct ‘crossflow’ solution, the initial guess for newton’s method is obtained by the ‘crossflow approximate’ solution; the function may not converge because of inaccuracy performing the numerical integral involved.

For the ‘crossflow, mixed 1&2’ solution, a bounded solver is first use, but the upper bound on P1 and the upper NTU1 limit is calculated from a pade approximation performed with mpmath.
Examples

>>> NTU_from_P_basic(P1=.975, R1=.1, subtype='counterflow')
3.984769850376482

ht.hx.NTU_from_P_J(P1, R1, Ntp)
Returns the number of transfer units of a TEMA J type heat exchanger with a specified (for side 1) thermal effectiveness \( P1 \), heat capacity ratio \( R1 \), and the number of tube passes \( Ntp \). The supported cases are as follows:

- One tube pass (shell fluid mixed)
- Two tube passes (shell fluid mixed, tube pass mixed between passes)
- Four tube passes (shell fluid mixed, tube pass mixed between passes)

Parameters

- \( P1 \) [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
- \( R1 \) [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- \( Ntp \) [int] Number of tube passes, 1, 2, or 4, [-]

Returns

- \( NTU1 \) [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

Notes

For numbers of tube passes that are not 1, 2, or 4, an exception is raised.

For the 1 tube pass case, a bounded solver is used to solve the equation numerically, with \( NTU1 \) ranging from 1E-11 to 1E3. \( NTU1 \) grows extremely quickly near its upper limit (\( NTU1 \) diverges to infinity at this maximum, but because the solver is bounded it will only increase up to 1000 before an exception is raised).

>>> NTU_from_P_J(P1=.995024, R1=.01, Ntp=1)
13.940758768266656
>>> NTU_from_P_J(P1=.99502487562189, R1=.01, Ntp=1)  # doctest: +SKIP
ValueError: No solution possible gives such a high P1; maximum P1=0.995025 at
\( \rightarrow NTU1=1000.000000 \)

For the 2 pass and 4 pass solution, a bounded solver is first use, but the upper bound on \( P1 \) and the upper \( NTU1 \) limit is calculated from a pade approximation performed with mpmath. These normally do not allow \( NTU1 \) to rise above 100.

Examples

>>> NTU_from_P_J(P1=.57, R1=1/3., Ntp=1)
1.0003070138879664

ht.hx.NTU_from_P_G(P1, R1, Ntp, optimal=True)
Returns the number of transfer units of a TEMA G type heat exchanger with a specified (for side 1) thermal effectiveness \( P1 \), heat capacity ratio \( R1 \), the number of tube passes \( Ntp \), and for the two-pass case whether or not the inlets are arranged optimally. The supported cases are as follows:
• One tube pass (tube fluid split into two streams individually mixed, shell fluid mixed)
• Two tube passes (shell and tube exchanger with shell and tube fluids mixed in each pass at the cross section), counterflow arrangement
• Two tube passes (shell and tube exchanger with shell and tube fluids mixed in each pass at the cross section), parallelflow arrangement

Parameters

**P1** [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

**R1** [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

**Ntp** [int] Number of tube passes, 1 or 2 [-]

**optimal** [bool, optional] Whether or not the arrangement is configured to give more of a countercurrent and efficient (True) case or an inefficient parallel case (only applies for two passes), [-]

Returns

**NTU1** [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

Notes

For numbers of tube passes greater than 1 or 2, an exception is raised.

Although this function allows the thermal effectiveness desired to be specified, it does not mean such a high value can be obtained. An exception is raised which shows the maximum possible effectiveness obtainable at the specified \(R1\) and configuration.

```python
>>> NTU_from_P_G(P1=1, R1=1/3., Ntp=2)
ValueError: No solution possible gives such a high P1; maximum P1=0.954545 at __NTU1=10000.000000
```

Of the three configurations, 1 pass and the optimal 2 pass have monotonic functions which allow for a bounded solver to work smoothly. In both cases a solution is searched for between NTU1 values of 1E-11 and 1E-4.

For the 2 pass unoptimal solution, a bounded solver is first use, but the upper bound on P1 and the upper NTU1 limit is calculated from a pade approximation performed with mpmath.

Examples

```python
>>> NTU_from_P_G(P1=.573, R1=1/3., Ntp=1)
0.9999513707759524
```

ht.hx.NTU_from_P_E(P1, R1, Ntp, optimal=True)

Returns the number of transfer units of a TEMA E type heat exchanger with a specified (for side 1) thermal effectiveness \(P1\), heat capacity ratio \(R1\), the number of tube passes \(Ntp\), and for the two-pass case whether or not the inlets are arranged optimally. The supported cases are as follows:

- 1-1 TEMA E, shell fluid mixed
- 1-2 TEMA E, shell fluid mixed (this configuration is symmetric)
• 1-2 TEMA E, shell fluid split into two streams individually mixed
• 1-3 TEMA E, shell and tube fluids mixed, one parallel pass and two countercflow passes (efficient)
• 1-3 TEMA E, shell and tube fluids mixed, two parallel passes and one countercflow pass (inefficient)
• 1-N TEMA E, shell and tube fluids mixed, efficient countercflow orientation, N an even number

Two of these cases have analytical solutions; the rest use numerical solvers of varying quality.

The analytical solution to 1-1 TEMA E, shell fluid mixed (the same as pure countercflow):

\[ NTU_1 = -\frac{1}{R_1 - 1} \log \left( \frac{P_1 R_1 - 1}{P_1 - 1} \right) \]

1-2 TEMA E, shell fluid mixed:

\[ NTU_1 = \frac{2}{\sqrt{R_1^2 + 1}} \log \left( \frac{P_1 R_1 - P_1 \sqrt{R_1^2 + 1} + P_1 - 2}{P_1 R_1 + P_1 \sqrt{R_1^2 + 1} + P_1 - 2} \right) \]

**Parameters**

- `P1` [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
- `R1` [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- `Ntp` [int] Number of tube passes, 1, 2, 3, 4, or an even number [-]
- `optimal` [bool, optional] Whether or not the arrangement is configured to give more of a countercurrent and efficient (True) case or an inefficient parallel case, [-]

**Returns**

- `NTU1` [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

**Notes**

For odd numbers of tube passes greater than 3, an exception is raised.

For the 2 pass, unoptimal case, a bounded solver is used with NTU1 between 1E-11 and 100; the solution to any feasible P1 was found to lie in there. For the 4 or a higher even number of pass case, the upper limit on NTU1 is 1000; this solver works pretty well, but as NTU1 reaches its limit the change in P1 is so small a smaller but also correct solution is often returned.

For both the optimal and unoptimal 3 tube pass case, a solution is only returned if NTU1 is between 1E-11 and 10. These functions are extremely mathematically frustrating, and as NTU1 rises above 10 catastrophic cancellation quickly results in this expression finding a ZeroDivisionError. The use of arbitrary prevision helps little - quickly 1000 digits are needed, and then 1000000 digits, and so one. Using SymPy's rational number support works better but is extremely slow for these complicated solutions. Nevertheless, so long as a solution is between 1E-11 and 10, the solver is quite robust.

**Examples**

```python
>>> NTU_from_P_E(P1=.58, R1=1/3., Ntp=2)
1.0381979240816719
```
ht.hx.NTU_from_P_H(P1, R1, Ntp, optimal=True)

Returns the number of transfer units of a TEMA H type heat exchanger with a specified (for side 1) thermal effectiveness \( P1 \), heat capacity ratio \( R1 \), the number of tube passes \( Ntp \), and for the two-pass case whether or not the inlets are arranged optimally. The supported cases are as follows:

- One tube pass (tube fluid split into two streams individually mixed, shell fluid mixed)
- Two tube passes (shell fluid mixed, tube pass mixed between passes)
- Two tube passes (shell fluid mixed, tube pass mixed between passes, inlet tube side next to inlet shell-side)

**Parameters**

- \( P1 \) [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
- \( R1 \) [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]
- \( Ntp \) [int] Number of tube passes, 1, or 2, [-]
- \( optimal \) [bool, optional] Whether or not the arrangement is configured to give more of a countercurrent and efficient (True) case or an inefficient parallel case, [-]

**Returns**

- \( NTU1 \) [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 (shell side = 1, tube side = 2) [-]

**Notes**

For numbers of tube passes greater than 1 or 2, an exception is raised.

Only numerical solutions are available for this function. For the case of 1 tube pass or the optimal 2 tube pass, the function is monotonic and a bounded solver is used with \( NTU1 \) between 1E-11 and 100; it will find the solution anywhere in that range.

For the non-optimal 2 pass case, the function is not monotonic and a pade approximation was used to obtain a curve of \( NTU1s \) which give the maximum \( P1s \) which is used as the upper bound in the bounded solver. The lower bound is still 1E-11. These solvers are all robust.

**Examples**

```python
>>> NTU_from_P_H(P1=0.573, R1=1/3., Ntp=1)
0.9997628696891168
```

ht.hx.NTU_from_P_plate(P1, R1, Np1, Np2, counterflow=True, passes_counterflow=True, reverse=False)

Returns the number of transfer units of a plate heat exchanger with a specified side 1 heat capacity ratio \( R1 \), side 1 number of transfer units \( NTU1 \), number of passes on sides 1 and 2 (respectively \( Np1 \) and \( Np2 \)).

For all cases, the function also takes as arguments whether the exchanger is setup in an overall counter or parallel orientation \( counterflow \), and whether or not individual stream passes are themselves counterflow or parallel.

The 20 supported cases are as follows. (the first number of sides listed refers to side 1, and the second number refers to side 2):

- 1 pass/1 pass parallelflow
- 1 pass/1 pass counterflow
• 1 pass/2 pass
• 1 pass/3 pass or 3 pass/1 pass (with the two end passes in parallel)
• 1 pass/3 pass or 3 pass/1 pass (with the two end passes in counterflow)
• 1 pass/4 pass
• 2 pass/2 pass, overall parallelflow, individual passes in parallel
• 2 pass/2 pass, overall parallelflow, individual passes counterflow
• 2 pass/2 pass, overall counterflow, individual passes parallelflow
• 2 pass/2 pass, overall counterflow, individual passes counterflow
• 2 pass/3 pass or 3 pass/2 pass, overall parallelflow
• 2 pass/3 pass or 3 pass/2 pass, overall counterflow
• 2 pass/4 pass or 4 pass/2 pass, overall parallel flow
• 2 pass/4 pass or 4 pass/2 pass, overall counterflow flow

For all except the simplest cases numerical solutions are used.

1 pass/1 pass counterflow (also 2/2 fully counterflow):

$$NTU_1 = -\frac{1}{R_1 - 1} \log \left( \frac{P_1 R_1 - 1}{P_1 - 1} \right)$$

1 pass/1 pass parallel flow (also 2/2 fully parallelflow):

$$NTU_1 = \frac{1}{R_1 + 1} \log \left( -\frac{1}{P_1 (R_1 + 1) - 1} \right)$$

Parameters

P1 [float] Thermal effectiveness of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

R1 [float] Heat capacity ratio of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]

Np1 [int] Number of passes on side 1 [-]

Np2 [int] Number of passes on side 2 [-]

counterflow [bool] Whether or not the overall flow through the heat exchanger is in counterflow or parallel flow, [-]

passes_counterflow [bool] In addition to the overall flow direction, in some cases individual passes may be in counter or parallel flow; this controls that [-]

reverse [bool] Used internally only to allow cases like the 1-4 formula to work for the 4-1 flow case, without having to duplicate the code [-]

Returns

NTU1 [float] Thermal number of transfer units of the heat exchanger in the P-NTU method, calculated with respect to stream 1 [-]
Notes

The defaults of counterflow=True and passes_counterflow=True will always result in the most efficient heat exchanger option, normally what is desired.

If a number of passes which is not supported is provided, an exception is raised.

For more details, see temperature_effectiveness_plate.

Examples

Three passes on side 1; one pass on side 2; two end passes in counterflow orientation.

```python
>>> NTU_from_P_plate(P1=0.5743, R1=1/3., Np1=3, Np2=1)
0.9998336056090733
```

```python
ht.hx.check_tubing_TEMA(NPS=None, BWG=None)
```

```python
>>> check_tubing_TEMA(2, 22)
False
>>> check_tubing_TEMA(0.375, 22)
True
```

```python
ht.hx.get_tube_TEMA(NPS=None, BWG=None, Do=None, Di=None, tmin=None)
```

```python
ht.hx.DBundle_min(Do)
```

Very roughly, determines a good choice of shell diameter for a given tube outer diameter, according to figure 1, section 3.3.5 in [1].

Parameters

- **Do** [float] Tube outer diameter, [m]

Returns

- **DShell** [float] Shell inner diameter, optional, [m]

Notes

This function should be used if a tube diameter is specified but not a shell size. DShell will have to be adjusted later, once the area requirement is known. This function is essentially a lookup table.

References

[1]

Examples

```python
>>> DBundle_min(0.0254)
1.0
```

```python
ht.hx.shell_clearance(DBundle=None, DShell=None)
```

Looks up the recommended clearance between a shell and tube bundle in a TEMA HX [1]. Either the bundle diameter or the shell diameter are needed provided.
Parameters

**DBundle** [float, optional] Outer diameter of tube bundle, [m]

**DShell** [float, optional] Shell inner diameter, [m]

Returns

**c** [float] Shell-tube bundle clearance, [m]

Notes

Lower limits are extended up to the next limit where intermediate limits are not provided.

References

[1]

Examples

```python
>>> shell_clearance(DBundle=1.245)
0.0064
```

**ht.hx.baffle_thickness**(*Dshell, L_unsupported, service='C')*

Determines the thickness of baffles and support plates in TEMA HX [1]. Applies to latitudinal baffles along the diameter of the HX, but not longitudinal baffles parallel to the tubes.

Parameters

**Dshell** [float] Shell inner diameter, [m]

**L_unsupported** [float] Distance between tube supports, [m]

**service** [str] Service type, C, R or B, [-]

Returns

**t** [float] Baffle or support plate thickness, [m]

Notes

No checks are implemented to ensure the given shell size is TEMA compatible. The baffles do not need to be strong as the pressure is almost the same on both of their sides. *L_unsupported* is a design choice; the more baffles in a given length, the higher the pressure drop.

References

[1]

Examples

```python
>>> baffle_thickness(Dshell=.3, L_unsupported=50, service='R')
0.0095
```
ht.hx.D_baffle_holes\((do, L\_unsupported)\)
Determine the diameter of holes in baffles for tubes according to TEMA [1]. Applies for all geometries.

Parameters

- **do** [float] Tube outer diameter, [m]
- **L\_unsupported** [float] Distance between tube supports, [m]

Returns

- **dB** [float] Baffle hole diameter, [m]

References

[1]

Examples

```python
>>> D_baffle_holes(do=0.0508, L_unsupported=0.75)
0.0516
>>> D_baffle_holes(do=0.01905, L_unsupported=0.3)
0.01985
>>> D_baffle_holes(do=0.01905, L_unsupported=1.5)
0.01945000000000002
```

ht.hx.L_unsupported_max\((Do, material='CS')\)
Determine the maximum length of a heat exchanger tube can go without a support, according to TEMA [1]. The limits provided apply for the worst-case temperature allowed for the material to be used at.

Parameters

- **Do** [float] Outer tube diameter, [m]
- **material** [str] Material type, either 'CS' or 'aluminium', [-]

Returns

- **L\_unsupported** [float] Maximum length of unsupported tube, [m]

Notes

The ‘CS’ results is also listed as representing high alloy steel, low alloy steel, nickel-copper, nickel, and nickel-chromium-iron alloys. The ‘aluminium’ results are those of copper and copper alloys and also titanium alloys. The maximum and minimum tube outer diameter tabulated are 3 inch and 1/4 inch respectively. The result is returned for the nearest tube diameter equal or smaller than the provided diameter, which helps ensures the returned tube length will not be optimistic. However, if the diameter is under 0.25 inches, the result will be optimistic!

References

[1]
Examples

```python
>>> L_unsupported_max(Do=.0254, material='CS')
1.88
```

ht.hx.Ntubes (DBundle, Do, pitch, Ntp=1, angle=30, Method=None)
Calculates the number of tubes which can fit in a heat exchanger. The tube count is effected by the pitch, number of tube passes, and angle.

The result is an exact number of tubes and is calculated by a very accurate method using number theory by default. This method is available only up to 100,000 tubes.

Parameters
- DBundle [float] Outer diameter of tube bundle, [m]
- Do [float] Tube outer diameter, [m]
- pitch [float] Pitch; distance between two orthogonal tube centers, [m]
- Ntp [int, optional] Number of tube passes, [-]
- angle [float, optional] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

Returns
- N [int] Total number of tubes that fit in the heat exchanger, [-]

Other Parameters

See also:
- Ntubes_Phadkeb
- Ntubes_Perrys
- Ntubes_VDI
- Ntubes_HEDH
- size_bundle_from_tubecount

Examples

```python
>>> Ntubes(DBundle=1.2, Do=0.025, pitch=0.03125)
1285
```

The approximations are pretty good too:

```python
>>> Ntubes(DBundle=1.2, Do=0.025, pitch=0.03125, Method='Perry')
1297
>>> Ntubes(DBundle=1.2, Do=0.025, pitch=0.03125, Method='VDI')
1340
>>> Ntubes(DBundle=1.2, Do=0.025, pitch=0.03125, Method='HEDH')
1272
```

ht.hx.size_bundle_from_tubecount (N, Do, pitch, Ntp=1, angle=30, Method=None)
Calculates the outer diameter of a tube bundle containing a specified number of tubes. The tube count is effected by the pitch, number of tube passes, and angle.
The result is an exact number of tubes and is calculated by a very accurate method using number theory by default. This method is available only up to 100,000 tubes.

**Parameters**

- \( N \) [int] Total number of tubes that fit in the heat exchanger, [-]
- \( D_0 \) [float] Tube outer diameter, [m]
- \( \text{pitch} \) [float] Pitch; distance between two orthogonal tube centers, [m]
- \( N_{tp} \) [int, optional] Number of tube passes, [-]
- \( \text{angle} \) [float, optional] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

**Returns**

- \( DBundle \) [float] Outer diameter of tube bundle, [m]

**Other Parameters**

- **Method** [string, optional] One of ‘Phadkeb’, ‘HEDH’, ‘VDI’ or ‘Perry’

**See also:**

- \( N_{tubes} \)
- \( DBundle \_for\_N_{tubes}\_Phadkeb \)
- \( D \_for\_N_{tubes}\_VDI \)
- \( DBundle \_for\_N_{tubes}\_HEDH \)

**Notes**

The ‘Perry’ method is solved with a numerical solver and is very unreliable.

**Examples**

```python
>>> size_bundle_from_tubecount(N=1285, Do=0.025, pitch=0.03125)
1.1985676402390355
```

ht.hx.\texttt{Ntubes\_Perrys}\(\texttt{(DBundle, Do, Ntp, angle=30)}\)


**Parameters**

- \( DBundle \) [float] Outer diameter of tube bundle, [m]
- \( Do \) [int] Tube outer diameter, [m]
- \( N_{tp} \) [float] Number of tube passes, [-]
- \( \text{angle} \) [float] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

**Returns**

- \( Nt \) [int] Number of tubes, [-]
Notes

Perry’s equation 11-74. Pitch equal to 1.25 times the tube outside diameter No other source for this equation is given. Experience suggests this is accurate to 40 tubes, but is often around 20 tubes off.

References

[1]

Examples

```python
>>> Ntubes_Perrys(DBundle=1.184, Do=.028, Ntp=2, angle=45)
803
```

A rough equation presented in the VDI Heat Atlas for estimating the number of tubes in a tube bundle of differing geometries and tube sizes. No accuracy estimation given.

Parameters

- **DBundle** [float] Outer diameter of tube bundle, [m]
- **Ntp** [float] Number of tube passes, [-]
- **Do** [float] Tube outer diameter, [m]
- **pitch** [float] Pitch; distance between two orthogonal tube centers, [m]
- **angle** [float] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

Returns

- **N** [float] Number of tubes, [-]

Notes

No coefficients for this method with Ntp=6 are available in [1]. For consistency, estimated values were added to support 6 tube passes, \( f_2 = 90 \). This equation is a rearranged form of that presented in [1]. The calculated tube count is rounded down to an integer.

References

[1]

Examples

```python
>>> Ntubes_VDI(DBundle=None, Ntp=None, Do=None, pitch=None, angle=30.0)
```

A rough equation presented in the VDI Heat Atlas for estimating the number of tubes in a tube bundle of differing geometries and tube sizes. No accuracy estimation given.

Parameters

- **DBundle** [float] Outer diameter of tube bundle, [m]
- **Ntp** [float] Number of tube passes, [-]
- **Do** [float] Tube outer diameter, [m]
- **pitch** [float] Pitch; distance between two orthogonal tube centers, [m]
- **angle** [float] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

Returns

- **N** [float] Number of tubes, [-]

Notes

No coefficients for this method with Ntp=6 are available in [1]. For consistency, estimated values were added to support 6 tube passes, \( f_2 = 90 \). This equation is a rearranged form of that presented in [1]. The calculated tube count is rounded down to an integer.

References

[1]

Examples

```python
>>> Ntubes_VDI(DBundle=1.184, Ntp=2, Do=.028, pitch=.036, angle=30)
966
```

Using tabulated values and correction factors for number of passes, the highly accurate method of [1] is used to obtain the tube count of a given tube bundle outer diameter for a given tube size and pitch.

Parameters
DBundle [float] Outer diameter of tube bundle, [m]
Do [float] Tube outer diameter, [m]
pitch [float] Pitch; distance between two orthogonal tube centers, [m]
Ntp [int] Number of tube passes, [-]
angle [float, optional] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

Returns

Nt [int] Total number of tubes that fit in the heat exchanger, [-]

Notes

For single-pass cases, the result is exact, and no tubes need to be removed for any reason. For 4, 6, 8 pass arrangements, a number of tubes must be removed to accommodate pass partition plates. The following assumptions are involved with that:

- The pass partition plate is where a row of tubes would have been. Only one or two rows are assumed affected.
- The thickness of partition plate is < 70% of the tube outer diameter.
- The distance between the centerline of the partition plate and the centerline of the nearest row of tubes is equal to the pitch.

This function will fail when there are more than 100,000 tubes. [1] tabulated values up to approximately 3,000 tubes derived with number theory. The sequences of integers were identified in the On-Line Encyclopedia of Integer Sequences (OEIS), and formulas listed in it were used to generate more coefficient to allow up to 100,000 tubes. The integer sequences are A003136, A038590, A001481, and A057961. The generation of coefficients for A038590 is very slow, but the rest are reasonably fast.

The number of tubes that fit generally does not increase one-by-one, but by several.

```python
>>> Ntubes_Phadkeb(DBundle=1.007, Do=.028, pitch=.036, Ntp=2, angle=45.)
558
>>> Ntubes_Phadkeb(DBundle=1.008, Do=.028, pitch=.036, Ntp=2, angle=45.)
574
```

Because a pass partition needs to be installed in multiple tube pass shells, more tubes fit in an exchanger the fewer passes are used.

```python
>>> Ntubes_Phadkeb(DBundle=1.008, Do=.028, pitch=.036, Ntp=1, angle=45.)
593
```

References

[1]

Examples

```python
>>> Ntubes_Phadkeb(DBundle=1.200-.008*2, Do=.028, pitch=.036, Ntp=2, angle=45.)
782
```
ht.hx.**DBundle_for_Ntubes_Phade** (Ntubes, Do, pitch, Ntp, angle=30)
Determine the bundle diameter required to fit a specified number of tubes in a heat exchanger. Uses the highly
accurate method of [1], which takes into account pitch, number of tube passes, angle, and tube diameter. The
method is analytically correct when used in the other direction (calculating number of tubes from bundle diam-
eter); in reverse, it is solved by bisection.

**Parameters**

- **Ntubes** [int] Total number of tubes that fit in the heat exchanger, [-]
- **Do** [float] Tube outer diameter, [m]
- **pitch** [float] Pitch; distance between two orthogonal tube centers, [m]
- **Ntp** [int] Number of tube passes, [-]
- **angle** [float, optional] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

**Returns**

- **DBundle** [float] Outer diameter of tube bundle, [m]

**Notes**

This function will fail when there are more than 100,000 tubes. There are a range of correct diameters for which
there can be the given number of tubes; a number within that range is returned as found by bisection.

**References**

[1]

**Examples**

```python
>>> DBundle_for_Ntubes_Phadeb(Ntubes=782, Do=.028, pitch=.036, Ntp=2, angle=45.)
1.1879392959379533
```

ht.hx.**Ntubes_HEDH** (DBundle=None, Do=None, pitch=None, angle=30)
A rough equation presented in the HEDH for estimating the number of tubes in a tube bundle of differing
geometries and tube sizes. No accuracy estimation given. Only 1 pass is supported.

\[
N = \frac{0.78(D_{bundle} - D_o)^2}{C_1(pitch)^2}
\]

C1 = 0.866 for 30° and 60° layouts, and 1 for 45 and 90° layouts.

**Parameters**

- **DBundle** [float] Outer diameter of tube bundle, [m]
- **Do** [float] Tube outer diameter, [m]
- **pitch** [float] Pitch; distance between two orthogonal tube centers, [m]
- **angle** [float] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

**Returns**

- **N** [float] Number of tubes, [-]
Notes

Seems reasonably accurate.

References

[1]

Examples

```python
>>> Ntubes_HEDH(DBundle=1.184, Do=.028, pitch=.036, angle=30)
928
```

ht.hx.DBundle_for_Ntubes_HEDH(N, Do, pitch, angle=30)
A rough equation presented in the HEDH for estimating the tube bundle diameter necessary to fit a given number of tubes. No accuracy estimation given. Only 1 pass is supported.

\[
D_{bundle} = (D_o + (pitch) \sqrt{\frac{1}{0.78} \cdot \sqrt{C_1 \cdot N}})
\]

C1 = 0.866 for 30° and 60° layouts, and 1 for 45 and 90° layouts.

Parameters

- **N** [float] Number of tubes, [-]
- **Do** [float] Tube outer diameter, [m]
- **pitch** [float] Pitch; distance between two orthogonal tube centers, [m]
- **angle** [float] The angle the tubes are positioned; 30, 45, 60 or 90, [degrees]

Returns

- **DBundle** [float] Outer diameter of tube bundle, [m]

Notes

Easily reversed from the main formulation.

References

[1]

Examples

```python
>>> DBundle_for_Ntubes_HEDH(N=928, Do=.028, pitch=.036, angle=30)
1.184
```

ht.hx.D_for_Ntubes_VDI(N, Ntp, Do, pitch, angle=30)
A rough equation presented in the VDI Heat Atlas for estimating the size of a tube bundle from a given number of tubes, number of tube passes, outer tube diameter, pitch, and arrangement. No accuracy estimation given.

\[
O TL = \sqrt{f_1 z t^2 + f_2 z^2} - d_o
\]
Parameters

- **N** [float] Number of tubes, [-]
- **Ntp** [float] Number of tube passes, [-]
- **Do** [float] Tube outer diameter, [m]
- **pitch** [float] Pitch; distance between two orthogonal tube centers, [m]
- **angle** [float] The angle the tubes are positioned; 30, 45, 60 or 90

Returns

- **DBundle** [float] Outer diameter of tube bundle, [m]

Notes

f1 = 1.1 for triangular, 1.3 for square patterns f2 is as follows: 1 pass, 0; 2 passes, 22; 4 passes, 70; 8 passes, 105. 6 tube passes is not officially supported, only 1, 2, 4 and 8. However, an estimated constant has been added to support it. f2 = 90.

References

[1]

Examples

```python
>>> D_for_Ntubes_VDI(N=970, Ntp=2., Do=0.00735, pitch=0.015, angle=30.)
0.5003600119829544
```

2.19 Database of insulating and refractory material properties (ht.insulation)

ht.insulation.nearest_material(name, complete=False)

Returns the nearest hit to a given name from from dictionaries of building, insulating, or refractory material from tables in [1], [2], and [3]. Function will pick the closest match based on a fuzzy search. if complete is True, will only return hits with all three of density, heat capacity, and thermal conductivity available.

Parameters

- **name** [str] Search keywords to be used by difflib function
- **complete** [bool, optional] If True, returns only hits with all parameters available

Returns

- **ID** [str] A key to one of the dictionaries mentioned above

References

[1], [2], [3]
Examples

```python
>>> nearest_material('stainless steel')
'Metals, stainless steel'
```

**ht.insulation.**\texttt{k\_material(\textit{ID}, \textit{T}=298.15)}

Returns thermal conductivity of a building, insulating, or refractory material from tables in [1], [2], and [3]. Thermal conductivity may or may not be dependent on temperature depending on the source used. Function must be provided with either a key to one of the dictionaries \texttt{refractories, ASHRAE, or building\_materials} - or a search term which will pick the closest match based on a fuzzy search. To determine which source the fuzzy search will pick, use the function \texttt{nearest\_material}. Fuzzy searches are slow; it is preferable to call this function with a material key directly.

**Parameters**

- \textbf{ID} [str] String as described above
- \textbf{T} [float, optional] Temperature of the material, [K]

**Returns**

- \textbf{k} [float] Thermal conductivity of the material, [W/m/K]

**References**

[1], [2], [3]

**Examples**

```python
>>> k_material('Mineral fiber')
0.036
```

**ht.insulation.**\texttt{rho\_material(\textit{ID})}

Returns the density of a building, insulating, or refractory material from tables in [1], [2], and [3]. No temperature dependence is available. Function must be provided with either a key to one of the dictionaries \texttt{refractories, ASHRAE, or building\_materials} - or a search term which will pick the closest match based on a fuzzy search. To determine which source the fuzzy search will pick, use the function \texttt{nearest\_material}. Fuzzy searches are slow; it is preferable to call this function with a material key directly.

**Parameters**

- \textbf{ID} [str] String as described above

**Returns**

- \textbf{rho} [float] Density of the material, [kg/m^3]

**References**

[1], [2], [3]

**Examples**

```python
>>> rho_material('Board, Asbestos/cement')
1900.0
```
ht.insulation.Cp_material (ID, T=298.15)

Returns heat capacity of a building, insulating, or refractory material from tables in [1], [2], and [3]. Heat capacity may or may not be dependent on temperature depending on the source used. Function must be provided with either a key to one of the dictionaries refractories, ASHRAE, or building_materials - or a search term which will pick the closest match based on a fuzzy search. To determine which source the fuzzy search will pick, use the function nearest_material. Fuzzy searches are slow; it is preferable to call this function with a material key directly.

Parameters

ID [str] String as described above
T [float, optional] Temperature of the material, [K]

Returns

Cp [float] Heat capacity of the material, [W/m/K]

References

[1], [2], [3]

Examples

>>> Cp_material('Mineral fiber')
840.0

ht.insulation.ASHRAE_k (ID)

Returns thermal conductivity of a building or insulating material from a table in [1]. Thermal conductivity is independent of temperature here. Many entries in the table are listed for varying densities, but the appropriate ID from the table must be selected to account for that.

Parameters

ID [str] ID corresponding to a material in the dictionary ASHRAE

Returns

k [float] Thermal conductivity of the material, [W/m/K]

References

[1]

Examples

>>> ASHRAE_k(ID='Mineral fiber')
0.036

ht.insulation.refractory_VDI_k (ID, T=None)

Returns thermal conductivity of a refractory material from a table in [1]. Here, thermal conductivity is a function of temperature between 673.15 K and 1473.15 K according to linear interpolation among 5 equally-spaced points. Here, thermal conductivity is not a function of porosity, which can affect it. If T is outside the acceptable range, it is rounded to the nearest limit. If T is not provided, the lowest temperature’s value is provided.

Parameters

ID [str] ID corresponding to a material in the dictionary refractories
T [float] Temperature of the material, [K]

Returns

k [float] Thermal conductivity of the material, [W/m/K]
**Heat Transfer Documentation, Release 0.1.56**

**ID** [str] ID corresponding to a material in the dictionary *refractories*

**T** [float, optional] Temperature of the refractory material, [K]

**Returns**

**k** [float] Thermal conductivity of the refractory material, [W/m/K]

**References**

[1]

**Examples**

```python
>>> [refractory_VDI_k('Fused silica', i) for i in [None, 200, 1000, 1500]]
[1.44, 1.44, 1.58074, 1.73]
```

**ht.insulation**.**refractory_VDI_Cp**(ID, T=None)

Returns heat capacity of a refractory material from a table in [1]. Here, heat capacity is a function of temperature between 673.15 K and 1473.15 K according to linear interpolation among 5 equally-spaced points. Here, heat capacity is not a function of porosity, affects it. If T is outside the acceptable range, it is rounded to the nearest limit. If T is not provided, the lowest temperature’s value is provided.

**Parameters**

**ID** [str] ID corresponding to a material in the dictionary *refractories*

**T** [float, optional] Temperature of the refractory material, [K]

**Returns**

**Cp** [float] Heat capacity of the refractory material, [W/m/K]

**References**

[1]

**Examples**

```python
>>> [refractory_VDI_Cp('Fused silica', i) for i in [None, 200, 1000, 1500]]
[917.0, 917.0, 956.78225, 982.0]
```

**2.20 Support for Numba (ht.numba)**

Basic module which wraps most of *ht* functions and classes to be compatible with the Numba dynamic Python compiler. Numba is only supported on Python 3, and may require the latest version of Numba. Numba is rapidly evolving, and hopefully in the future it will support more of the functionality of *ht*.

Using the numba-accelerated version of *ht* is easy; simply call functions and classes from the *ht.numba* namespace. The *ht.numba* module must be imported separately; it is not loaded automatically as part of *ht*.
There is a delay while the code is compiled when using Numba; the speed is not quite free. Most, but not all compilations can be cached to save time in future loadings.

It is easy to compare the speed of a function with and without Numba.

```python
%timeit ht.numba.Ft_aircooler(Thi=125., Tho=45., Tci=25., Tco=95., Ntp=1, rows=4)
1.22 µs ± 41.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

%timeit ht.Ft_aircooler(Thi=125., Tho=45., Tci=25., Tco=95., Ntp=1, rows=4)
5.89 µs ± 274 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Not everything is faster in the numba interface. It is advisable to check that numba is indeed faster for your use case. Functions which take strings as inputs are also known to normally get slower; the numerical stuff is still being sped up but the string handling is slow:

```python
%timeit ht.numba.baffle_correction_Bell(0.82, method='spline')
16.5 µs ± 538 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

%timeit ht.baffle_correction_Bell(0.82, method='spline')
15.6 µs ± 457 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Nevertheless, using the function from the numba interface may be preferably, to allow an even larger program to be completely compiled in njit mode.

Today, the list of things known not to work is as follows:

- `dP_Zukauskas()` (needs some spline work)
- `cylindrical_heat_transfer()` (returns dictionaries)
- `effectiveness_NTU_method()` (returns dictionaries)
- `P_NTU_method()` (returns dictionaries)
- `NTU_from_effectiveness()` (does string-to-int conversion)
- `DBundle_min()` and `shell_clearance()` (needs work)
- `wall_factor_Nu()` and `wall_factor_fd()` (dictionary lookups)
- `solar_spectrum()` (external file reading)
- Everything in `ht.insulation`

### 2.20.1 Numpy Support

Numba also allows ht to provide any of its supported functions as a numpy universal function. Numpy’s wonderful broadcasting is implemented, so some arguments can be arrays and some can not.
Unfortunately, keyword-arguments are not supported by Numba. Also default arguments are not presently supported by Numba.

Despite these limitations is is here that Numba really shines! Arrays are Numba’s strength.

Please note this interface is provided, but what works and what doesn’t is mostly up to the numba project. This backend is not quite as polished as their normal engine.

## 2.21 Heat transfer by radiation (ht.radiation)

### ht.radiation.blackbody_spectral_radiance (T, wavelength)

Returns the spectral radiance, in units of W/m^2/sr/µm.

\[
I_{\lambda, \text{blackbody}}(\lambda, T) = \frac{2hc^2}{\lambda^5[\exp(hc/\lambda kT) - 1]}
\]

**Parameters**

- **T** [float] Temperature of the surface, [K]
- **wavelength** [float] Length of the wave to be considered, [m]

**Returns**

- **I** [float] Spectral radiance [W/(m^2*sr*m)]

**Notes**

Can be used to derive the Stefan-Boltzman law, or determine the maximum radiant frequency for a given temperature.

**References**

[1], [2]

**Examples**

Checked with Spectral-calc.com, at [2].

```python
>>> blackbody_spectral_radiance(800., 4E-6)
1311694129.7430933
```

Calculation of power from the sun (earth occupies 6.8E-5 steradian of the sun):

```python
>>> from scipy.integrate import quad
>>> rad = lambda l: blackbody_spectral_radiance(5778., l)*6.8E-5
>>> quad(rad, 1E-10, 1E-4)[0]
1367.9827067638964
```

### ht.radiation.q_rad (emissivity, T, T2=0)

Returns the radiant heat flux of a surface, optionally including assuming radiant heat transfer back to the surface.

\[ q = \epsilon\sigma(T_1^4 - T_2^4) \]

**Parameters**
emissivity  [float] Fraction of black-body radiation which is emitted, [-]
T   [float] Temperature of the surface, [K]
T2  [float, optional] Temperature of the surrounding material of the surface [K]

Returns
q   [float] Heat exchange [W/m^2]

Notes
Emissivity must be less than 1. T2 may be larger than T.

References
[1]

Examples
>>> q_rad(emissivity=1, T=400)
1451.613952

>>> q_rad(.85, T=400, T2=305.)
816.7821722650002

ht.radiation.grey_transmittance(extinction_coefficient, molar_density, length, base=2.718281828459045)
Calculates the transmittance of a grey body, given the extinction coefficient of the material, its molar density, and the path length of the radiation.

\[ \tau = \text{base}^{-e \cdot l \cdot \rho_m} \]

Parameters

extinction_coefficient  [float] The extinction coefficient of the material the radiation is passing at the modeled frequency, [m^2/mol]
molar_density  [float] The molar density of the material the radiation is passing through, [mol/m^3]
length  [float] The length of the body the radiation is transmitted through, [m]
base  [float, optional] The exponent used in calculations; e is more theoretically sound but 10 is often used as a base by chemists, [-]

Returns
transmittance  [float] The fraction of spectral radiance which is transmitted through a grey body (can be liquid, gas, or even solid ex. in the case of glasses) [-]

Notes
For extinction coefficients, see the HITRAN database. They are temperature and pressure dependent for each chemical and phase.
References

[1], [2]

Examples

Overall transmission loss through 1 cm of precipitable water equivalent atmospheric water vapor at a frequency of 1.3 um [2]:

```plaintext
>>> grey_transmittance(3.8e-4, molar_density=55300, length=1e-2)
0.8104707721191062
```

ht.radiation.solar_spectrum(model='SOLAR-ISS')

Returns the solar spectrum of the sun according to the specified model. Only the ‘SOLAR-ISS’ model is supported.

Parameters

- `model` [str, optional] The model to use; ‘SOLAR-ISS’ is the only model available, [-]

Returns

- `wavelengths` [ndarray] The wavelengths of the solar spectra, [m]
- `SSI` [ndarray] The solar spectral irradiance of the sun, [W/(m^2*m)]
- `uncertainties` [ndarray] The estimated absolute uncertainty of the measured spectral irradiance of the sun, [W/(m^2*m)]

Notes

The power of the sun changes as the earth gets closer or further away.
In [1], the UV and VIS data come from observations in 2008; the IR comes from measurements made from 2010-2016. There is a further 28 W/m^2 for the 3 micrometer to 160 micrometer range, not included in this model. All data was corrected to a standard distance of one astronomical unit from the Sun, as is the resultant spectrum.

The variation of the spectrum as a function of distance from the sun should alter only the absolute magnitudes.


99.9% of the time this function takes is to read in the solar data from disk. This could be reduced by using pandas.

References

[1], [2]

Examples

```plaintext
>>> wavelengths, SSI, uncertainties = solar_spectrum()
```

Calculate the minimum and maximum values of the wavelengths (0.5 nm/3000nm) and SSI:
Integration - calculate the solar constant, in units of W/m² hitting earth’s atmosphere.

```python
>>> min(wavelengths), max(wavelengths), min(SSI), max(SSI)
(5e-10, 2.9999000000000003e-06, 1330.0, 2256817820.0)
```

```python
>>> import numpy as np
>>> np.trapz(SSI, wavelengths)
1344.802978238
```

## 2.22 Support for numpy arrays (ht.vectorized)

Basic module which wraps all ht functions with numpy's vectorize. All other object - dicts, classes, etc - are not wrapped. Supports star imports; so the same objects exported when importing from the main library will be imported from here.

```python
>>> from ht.vectorized import *
```

Inputs do not need to be numpy arrays; they can be any iterable:

```python
>>> import ht.vectorized
>>> ht.vectorized.LMTD([100, 101], 60., 30., 40.2)
array([43.20040929, 43.60182765])
```

Note that because this needs to import ht itself, ht.vectorized needs to be imported separately; the following will cause an error:

```python
>>> import ht
>>> ht.vectorized # doctest: +SKIP
```

The correct syntax is as follows:

```python
>>> import ht.vectorized # Necessary
>>> from ht.vectorized import * # May be used without first importing ht
```
Get the latest version of ht from https://pypi.python.org/pypi/ht/

If you have an installation of Python with pip, simple install it with:

$ pip install ht

To get the git version, run:

$ git clone git://github.com/CalebBell/ht.git
The latest development version of ht’s sources can be obtained at

https://github.com/CalebBell/ht
Bug reports

To report bugs, please use the ht’s Bug Tracker at:

https://github.com/CalebBell/ht/issues
License information

ht is MIT licensed. See LICENSE.txt for information on the terms & conditions for usage of this software, and a DISCLAIMER OF ALL WARRANTIES.

Although not required by the ht license, if it is convenient for you, please cite ht if used in your work. Please also consider contributing any changes you make back, such that they may be incorporated into the main library and all of us will benefit from them.
Citation

To cite ht in publications use:

Caleb Bell (2016–2020). ht: Heat transfer component of Chemical Engineering Design Library (ChEDL)
https://github.com/CalebBell/ht.
Indices and tables

- genindex
- modindex
- search


[1] Li, Wei, and Zan Wu. “A General Correlation for Evaporative Heat Transfer in Micro/mi


<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ht.air_cooler</td>
<td>9</td>
</tr>
<tr>
<td>ht.boiling_flow</td>
<td>20</td>
</tr>
<tr>
<td>ht.boiling_nucleic</td>
<td>30</td>
</tr>
<tr>
<td>ht.boiling_plate</td>
<td>44</td>
</tr>
<tr>
<td>ht.condensation</td>
<td>50</td>
</tr>
<tr>
<td>ht.conduction</td>
<td>55</td>
</tr>
<tr>
<td>ht.conv_external</td>
<td>64</td>
</tr>
<tr>
<td>ht.conv_free_enclosed</td>
<td>92</td>
</tr>
<tr>
<td>ht.conv_free_immersed</td>
<td>74</td>
</tr>
<tr>
<td>ht.conv_internal</td>
<td>98</td>
</tr>
<tr>
<td>ht.conv_jacket</td>
<td>119</td>
</tr>
<tr>
<td>ht.conv_packed_bed</td>
<td>123</td>
</tr>
<tr>
<td>ht.conv_plate</td>
<td>126</td>
</tr>
<tr>
<td>ht.conv_supercritical</td>
<td>129</td>
</tr>
<tr>
<td>ht.conv_tube_bank</td>
<td>144</td>
</tr>
<tr>
<td>ht.conv_two_phase</td>
<td>156</td>
</tr>
<tr>
<td>ht.core</td>
<td>166</td>
</tr>
<tr>
<td>ht.hx</td>
<td>171</td>
</tr>
<tr>
<td>ht.insulation</td>
<td>218</td>
</tr>
<tr>
<td>ht.radiation</td>
<td>223</td>
</tr>
</tbody>
</table>
Index

A
Aggour() (in module ht.conv_two_phase), 163
air_cooler_noise_GPSA() (in module ht.air_cooler), 10
air_cooler_noise_Mukherjee() (in module ht.air_cooler), 11
Akers_Deans_Crosser() (in module ht.condensation), 53
ASHRAE_k() (in module ht.insulation), 220

B
baffle_correction_Bell() (in module ht.conv_tube_bank), 152
baffle_leakage_Bell() (in module ht.conv_tube_bank), 153
baffle_thickness() (in module ht.hx), 210
Bier() (in module ht.boiling_nucleic), 36
blackbody_spectral_radiance() (in module ht.radiation), 223
Boyko_Kruzhilin() (in module ht.condensation), 50
bundle_bypassing_Bell() (in module ht.conv_tube_bank), 154

calc_Cmax() (in module ht.hx), 178
calc_Cmin() (in module ht.hx), 177
calc_Cr() (in module ht.hx), 179
Cavallini_Smith_Zecchin() (in module ht.condensation), 53
check_tubing_TEMA() (in module ht.hx), 209
Chen_Bennett() (in module ht.boiling_flow), 24
Chen_Edelstein() (in module ht.boiling_flow), 23
Cooper() (in module ht.boiling_nucleic), 37
Cp_material() (in module ht.insulation), 219
cylindrical_heat_transfer() (in module ht.conduction), 63

d
D_baffle_holes() (in module ht.hx), 210
D_for_Ntubes_VDI() (in module ht.hx), 217
Davis_David() (in module ht.conv_two_phase), 156
DBundle_for_Ntubes_HEDH() (in module ht.hx), 217
DBundle_for_Ntubes_Phadkeb() (in module ht.hx), 215
DBundle_min() (in module ht.hx), 209
dP_ESDU_high_fin() (in module ht.air_cooler), 17
dP_ESDU_low_fin() (in module ht.air_cooler), 18
dP_Kern() (in module ht.conv_tube_bank), 144
dP_Zukauskas() (in module ht.conv_tube_bank), 145

e
effectiveness_from_NTU() (in module ht.hx), 171
effectiveness_NTU_method() (in module ht.hx), 182
Elamvaluthi_Srinivas() (in module ht.conv_two_phase), 157
ESDU_tube_angle_correction() (in module ht.conv_tube_bank), 152
ESDU_tube_row_correction() (in module ht.conv_tube_bank), 151

f
F_LMTD_Fakheri() (in module ht.hx), 183
fin_efficiency_Kern_Kraus() (in module ht.core), 170
Forster_Zuber() (in module ht.boiling_nucleic), 32
Ft_aircooler() (in module ht.air_cooler), 9

g
get_tube_TEMA() (in module ht.hx), 209
Gorenflo() (in module ht.boiling_nucleic), 38
grey_transmittance() (in module ht.radiation), 224
Groothuis_Hendal() (in module ht.conv_two_phase), 158

269
H

h_boiling_Amalfi() (in module ht.boiling_plate), 44
h_boiling_Han_Lee_Kim() (in module ht.boiling_plate), 46
h_boiling_Huang_Sheer() (in module ht.boiling_plate), 48
h_boiling_Lee_Kang_Kim() (in module ht.boiling_plate), 45
h_boiling_Yan_Lin() (in module ht.boiling_plate), 49
h_Briggs_Young() (in module ht.air_cooler), 11
h_ESDU_high_fin() (in module ht.air_cooler), 12
h_ESDU_low_fin() (in module ht.air_cooler), 14
h_Ganguli_VDI() (in module ht.air_cooler), 16
h_kinetic() (in module ht.boiling_nucleic), 52
h_nucleic() (in module ht.boiling_nucleic), 39
h_nucleic_methods() (in module ht.boiling_nucleic), 40
h_two_phase() (in module ht.conv_two_phase), 164
h_two_phase_methods() (in module ht.conv_two_phase), 165
HEDH_Montinsky() (in module ht.boiling_nucleic), 42
HEDH_Taborek() (in module ht.boiling_nucleic), 35
helical_turbulent_Nu_Mori_Nakayama() (in module ht.conv_internal), 116
helical_turbulent_Nu_Schmidt() (in module ht.conv_internal), 117
helical_turbulent_Nu_Xin_Ebadian() (in module ht.conv_internal), 118
ht.air_cooler (module), 9
ht.boiling_flow (module), 20
ht.boiling_nucleic (module), 30
ht.boiling_plate (module), 44
ht.condensation (module), 50
ht.conduction (module), 55
ht.conv_external (module), 64
ht.conv_free_enclosed (module), 92
ht.conv_free_immersed (module), 74
ht.conv_internal (module), 98
ht.conv_jacket (module), 119
ht.conv_packed_bed (module), 123
ht.conv_plate (module), 126
ht.conv_supercritical (module), 129
ht.convTube_bank (module), 144
ht.conv_two_phase (module), 156
ht.core (module), 166
ht.hx (module), 171
ht.insulation (module), 218
ht.radiation (module), 223
Hughmark() (in module ht.conv_two_phase), 159

I

is_heating_property() (in module ht.core), 168
is_heating_temperature() (in module ht.core), 168

K

k_material() (in module ht.insulation), 219
k_to_R() (in module ht.conduction), 56
k_to_R_value() (in module ht.conduction), 58
k_to_thermal_resistivity() (in module ht.conduction), 56
Knott() (in module ht.conv_two_phase), 160
Kudirka_Grosh_McFadden() (in module ht.conv_two_phase), 161

L

L_unsupported_max() (in module ht.hx), 211
laminar_correction_Bell() (in module ht.conv_two_phase), 155
laminar_entry_Baehr_Stephan() (in module ht.Conv internal), 100
laminar_entry_Seider_Tate() (in module ht.conv_internal), 99
laminar_entry_thermal_Hausen() (in module ht.conv_internal), 99
laminar_Q_const() (in module ht.conv_internal), 98
laminar_T_const() (in module ht.conv_internal), 98
Lazarek_Black() (in module ht.boiling_flow), 26
Lehrer() (in module ht.conv_jacket), 119
Li_Wu() (in module ht.boiling_flow), 27
Liu_Winterton() (in module ht.boiling_flow), 22
LMTD() (in module ht.core), 166

M

Martin_Sims() (in module ht.conv_two_phase), 162
McNelly() (in module ht.boiling_nucleic), 31
Montinsky() (in module ht.boiling_nucleic), 33
Morimoto_Hotta() (in module ht.conv_internal), 115

N

nearest_material() (in module ht.insulation), 218
NTU_from_effectiveness() (in module ht.hx), 175
NTU_from_P_basic() (in module ht.hx), 202
NTU_from_P_E() (in module ht.hx), 205
NTU_from_P_G() (in module ht.hx), 204
NTU_from_P_H() (in module ht.hx), 206
NTU_from_P_J() (in module ht.hx), 204
NTU_from_P_plate() (in module ht.hx), 207
NTU_from_UA() (in module ht.hx), 181
Nu_plate_Kumar() (in module ht.conv_plate), 126
Nu_plate_Martin() (in module ht.conv_plate), 127
Nu_plate_Muley_Manglik() (in module ht.conv_plate), 127
Nu_Shitsman() (in module ht.conv_supercritical), 129
Nu_sphere_Churchill() (in module ht.conv_free_immersed), 79
Nu_Swenson() (in module ht.conv_supercritical), 133
Nu_vertical_cylinder() (in module ht.conv_free_immersed), 87
Nu_vertical_cylinder_Ali_Arabi_Khamis() (in module ht.conv_free_immersed), 86
Nu_vertical_cylinder_Carne_Morgan() (in module ht.conv_free_immersed), 81
Nu_vertical_cylinder_Eigenson_Morgan() (in module ht.conv_free_immersed), 82
Nu_vertical_cylinder_Griffiths_Davis_Morgan() (in module ht.conv_free_immersed), 80
Nu_vertical_cylinder_Hanesian_Kalish_Morgan() (in module ht.conv_free_immersed), 85
Nu_vertical_cylinder_Jakob_Linke_Morgan() (in module ht.conv_free_immersed), 81
Nu_vertical_cylinder_Kreith_Eckert() (in module ht.conv_free_immersed), 84
Nu_vertical_cylinder_McAdams_Weiss_Saunders() (in module ht.conv_free_immersed), 84
Nu_vertical_cylinder_methods() (in module ht.conv_free_immersed), 88
Nu_vertical_cylinder_Popiel_Churchill() (in module ht.conv_free_immersed), 86
Nu_vertical_cylinder_Touloukian_Morgan() (in module ht.conv_free_immersed), 83
Nu_vertical_helical_coil_Ali() (in module ht.conv_free_enclosed), 97
Nu_vertical_helical_coil_Prabhanjan_Rennie_Raghavan() (in module ht.conv_free_enclosed), 97
Nu_vertical_plate_Churchill() (in module ht.conv_free_immersed), 74
Nu_Wakao_Kagei() (in module ht.conv_packed_bed), 124
Nu_Xu() (in module ht.conv_supercritical), 134
Nu_Yamagata() (in module ht.conv_supercritical), 139
Nu_Zhu() (in module ht.conv_supercritical), 137
Nu_Zukauskas_Bejan() (in module ht.conv_tube_bank), 147
Nusselt_laminar() (in module ht.condensation), 51

Q
q_rad() (in module ht.radiation), 223
qmax_boiling() (in module ht.boiling_nucleic), 43
qmax_boiling_methods() (in module ht.boiling_nucleic), 44

R
R_cylinder() (in module ht.conduction), 58
R_to_k() (in module ht.conduction), 55
R_value_to_k() (in module ht.conduction), 57
Rac_Nusselt_Rayleigh() (in module ht.conv_free_enclosed), 94
Rac_Nusselt_Rayleigh_disk() (in module ht.conv_free_enclosed), 95
Ranpudi_Godbold() (in module ht.conv_free_enclosed), 162
refractory_VDI_Cp() (in module ht.insulation), 221
refractory_VDI_k() (in module ht.insulation), 220
rho_material() (in module ht.insulation), 219
Rohsenow() (in module ht.boiling_nucleic), 30

S
S_isothermal_pipe_eccentric_to_isothermal_pipe() (in module ht.conduction), 62
S_isothermal_pipe_normal_to_plane() (in module ht.conduction), 60
S_isothermal_pipe_to_isothermal_pipe() (in module ht.conduction), 61
S_isothermal_pipe_to_plan() (in module ht.conduction), 59
S_isothermal_pipe_to_two_planes() (in module ht.conduction), 62
S_isothermal_sphere_to_plan() (in module ht.conduction), 59
Serth_HEDH() (in module ht.boiling_nucleic), 42
Shah() (in module ht.condensation), 54
shell_clearance() (in module ht.hx), 209
size_bundle_from_tubecount() (in module ht.hx), 212
solar_spectrum() (in module ht.radiation), 225
Stein_Schmidt() (in module ht.conv_jacket), 121
Stephan_Abdelsalam() (in module ht.boiling_nucleic), 33
Sun_Mishima() (in module ht.boiling_flow), 28

temperature_effectiveness_air_cooler() (in module ht.hx), 196
temperature_effectiveness_basic() (in module ht.hx), 184
temperature_effectiveness_plate() (in module ht.hx), 192

P
P_NTU_method() (in module ht.hx), 198
Pc() (in module ht.hx), 180
Pp() (in module ht.hx), 179
temperature_effectiveness_TEMA_E() (in module ht.hx), 190

temperature_effectiveness_TEMA_G() (in module ht.hx), 189

temperature_effectiveness_TEMA_H() (in module ht.hx), 187

temperature_effectiveness_TEMA_J() (in module ht.hx), 186

thermal_resistivity_to_k() (in module ht.conduction), 57

Thome() (in module ht.boiling_flow), 20

turbulent_Bhatti_Shah() (in module ht.conv_internal), 113

turbulent_Churchill_Zajic() (in module ht.conv_internal), 109

turbulent_Colburn() (in module ht.conv_internal), 103

turbulent_Dipprey_Sabersky() (in module ht.conv_internal), 111

turbulent_Dittus_Boelter() (in module ht.conv_internal), 101

turbulent_Drexel_McAdams() (in module ht.conv_internal), 103

turbulent_entry_Hausen() (in module ht.conv_internal), 102

turbulent_ESDU() (in module ht.conv_internal), 109

turbulent_Friend_Metzner() (in module ht.conv_internal), 105

turbulent_Gnielinski() (in module ht.conv_internal), 107

turbulent_Gnielinski_smooth_1() (in module ht.conv_internal), 107

turbulent_Gnielinski_smooth_2() (in module ht.conv_internal), 108

turbulent_Gowen_Smith() (in module ht.conv_internal), 112

turbulent_Kawase_De() (in module ht.conv_internal), 113

turbulent_Kawase_Ulbrecht() (in module ht.conv_internal), 112

turbulent_Martinelli() (in module ht.conv_internal), 110

turbulent_Nunner() (in module ht.conv_internal), 110

turbulent_Petukhov_Kirillov_Popov() (in module ht.conv_internal), 105

turbulent_Prandtl() (in module ht.conv_internal), 104

turbulent_Sandall() (in module ht.conv_internal), 106

turbulent_Sieder_Tate() (in module ht.conv_internal), 101

turbulent_von_Karman() (in module ht.conv_internal), 106

U

UA_from_NTU() (in module ht.hx), 181

turbulent_Webb() (in module ht.conv_internal), 104

W

wall_factor() (in module ht.core), 167

wall_factor_fd() (in module ht.core), 169

wall_factor_Nu() (in module ht.core), 169

Y

Yun_Heo_Kim() (in module ht.boiling_flow), 29

Z

Zuber() (in module ht.boiling_nucleic), 41

Zukauskas_tube_row_correction() (in module ht.conv_tube_bank), 150

Index 273