Fluids Documentation

Release 0.1

Caleb Bell

April 02, 2016

Contents

Contents:

fluids

1.1 fluids package

1.1.1 Submodules

fluids.compressible module

Chemical Engineering Design Library (ChEDL). Utilities for process modeling. Copyright (C) 2016, Caleb Bell <Caleb.Andrew.Bell@gmail.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

`fluids.compressible.`**`T_critical_flow`**(*T*, *k*)

Calculates critical flow temperature *Tcf* for a fluid with the given isentropic coefficient. *Tcf* is in a flow (with Ma=1) whose stagnation conditions are known. Normally used with converging/diverging nozzles.

$$\frac{T^*}{T_0} = \frac{2}{k+1}$$

Parameters **T** : float

Stagnation temperature of a fluid with Ma=1 [K]

k : float

Isentropic coefficient []

Returns **Tcf** : float

Critical flow temperature at Ma=1 [K]

Notes

Assumes isentropic flow.

References

[R200]

Examples

Example 12.4 in *[R200]*:

```
>>> T_critical_flow(473, 1.289)
413.2809086937528
```

fluids.compressible.**P_critical_flow**(*P*, *k*)

Calculates critical flow pressure *Pcf* for a fluid with the given isentropic coefficient. *Pcf* is in a flow (with Ma=1) whose stagnation conditions are known. Normally used with converging/diverging nozzles.

$$\frac{P^*}{P_0} = \left(\frac{2}{k+1}\right)^{k/(k-1)}$$

Parameters **P** : float

Stagnation pressure of a fluid with Ma=1 [Pa]

k : float

Isentropic coefficient []

Returns **Pcf** : float

Critical flow pressure at Ma=1 [Pa]

Notes

Assumes isentropic flow.

References

[R201]

Examples

Example 12.4 in *[R201]*:

```
>>> P_critical_flow(1400000, 1.289)
766812.9022792266
```

fluids.compressible.**is_critical_flow**(*P1*, *P2*, *k*)

Determines if a flow of a fluid driven by pressure gradient P1 - P2 is critical, for a fluid with the given isentropic coefficient. This function calculates critical flow pressure, and checks if this is larger than P2. If so, the flow is critical and choked.

Parameters **P1: float**

Higher, source pressure [Pa]

P2: float

Lower, downstream pressure [Pa]

k : float

Isentropic coefficient []

Returns **flowtype** : bool

True if the flow is choked; otherwise False

Notes

Assumes isentropic flow. Uses P_critical_flow function.

References

[R202]

Examples

Examples 1-2 from API 520.

```
>>> is_critical_flow(670E3, 532E3, 1.11)
False
>>> is_critical_flow(670E3, 101E3, 1.11)
True
```

fluids.compressible.**stagnation_energy**(*V*)

Calculates the increase in enthalpy *dH* which is provided by a fluid's velocity *V*.

$$\Delta H = \frac{V^2}{2}$$

Parameters **V** : float

Velocity [m/s]

Returns **dH** : float

Incease in enthalpy [J/kg]

Notes

The units work out. This term is pretty small, but not trivial.

References

[R203]

Examples

```
>>> stagnation_energy(125)
7812.5
```

fluids.compressible.**P_stagnation**(*P*, *T*, *Tst*, *k*)

Calculates stagnation flow pressure *Pst* for a fluid with the given isentropic coefficient and specified stagnation temperature and normal temperature. Normally used with converging/diverging nozzles.

$$\frac{P_0}{P} = \left(\frac{T_0}{T}\right)^{\frac{k}{k-1}}$$

Parameters **P** : float

Normal pressure of a fluid [Pa]

T : float

Normal temperature of a fluid [K]

Tst : float

Stagnation temperature of a fluid moving at a certain velocity [K]

k : float

Isentropic coefficient []

Returns **Pst** : float

Stagnation pressure of a fluid moving at a certain velocity [Pa]

Notes

Assumes isentropic flow.

References

[R204]

Examples

Example 12-1 in *[R204]*.

```
>>> P_stagnation(54050., 255.7, 286.8, 1.4)
80772.80495900588
```

fluids.compressible.**T_stagnation**(*T*, *P*, *Pst*, *k*)

Calculates stagnation flow temperature *Tst* for a fluid with the given isentropic coefficient and specified stagnation pressure and normal pressure. Normally used with converging/diverging nozzles.

$$T = T_0 \left(\frac{P}{P_0}\right)^{\frac{k-1}{k}}$$

Parameters **T** : float

Normal temperature of a fluid [K]

P : float

Normal pressure of a fluid [Pa]

Pst : float

Stagnation pressure of a fluid moving at a certain velocity [Pa]

k : float

Isentropic coefficient []

Returns **Tst** : float

Stagnation temperature of a fluid moving at a certain velocity [K]

Notes

Assumes isentropic flow.

References

[R205]

Examples

Example 12-1 in *[R205]*.

```
>>> T_stagnation(286.8, 54050, 54050*8, 1.4)
519.5230938217768
```

fluids.compressible.**T_stagnation_ideal**(*T*, *V*, *Cp*)

Calculates the ideal stagnation temperature *Tst* calculated assuming the fluid has a constant heat capacity *Cp* and with a specified velocity *V* and tempeature *T*.

$$T^* = T + \frac{V^2}{2C_p}$$

Parameters **T** : float

Tempearture [K]

V : float

Velocity [m/s]

Cp : float

Ideal heat capacity [J/kg/K]

Returns **Tst** : float

Stagnation temperature [J/kg]

References

[R206]

Examples

Example 12-1 in *[R206]*.

```
>>> T_stagnation_ideal(255.7, 250, 1005.)
286.79452736318405
```

fluids.control_valve module

Chemical Engineering Design Library (ChEDL). Utilities for process modeling. Copyright (C) 2016, Caleb Bell <Caleb.Andrew.Bell@gmail.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

`fluids.control_valve.`**`cavitation_index`**(*P1*, *P2*, *Psat*)

Calculates the cavitation index of a valve with upstream and downstream absolute pressures *P1* and *P2* for a fluid with a vapor pressure *Psat*.

$$\sigma = \frac{P_1 - P_{sat}}{P_1 - P_2}$$

Parameters **P1** : float

Absolute pressure upstream of the valve [Pa]

P2 : float

Absolute pressure downstream of the valve [Pa]

Psat : float

Saturation pressure of the liquid at inlet temperature [Pa]

Returns **sigma** : float

Cavitation index of the valve [-]

Notes

Larger values are safer. Models for adjusting cavitation indexes provided by the manufacturer to the user's conditions are available, making use of scaling the pressure differences and size differences.

Values can be calculated for incipient cavitation, constant cavitation, maximum vibration cavitation, incipient damage, and choking cavitation.

Has also been defined as:

$$\sigma = \frac{P_2 - P_{sat}}{P_1 - P_2}$$

Another definition and notation series is:

$$K = xF = \frac{1}{\sigma} = \frac{P_1 - P_2}{P_1 - P_{sat}}$$

References

[R207]

Examples

```
>>> cavitation_index(1E6, 8E5, 2E5)
4.0
```

fluids.control_valve.**size_control_valve_l**(*rho*, *Psat*, *Pc*, *mu*, *P1*, *P2*, *Q*, *D1*, *D2*, *d*, *FL*, *Fd*)

Calculates flow coefficient of a control valve passing a liquid according to IEC 60534. Uses a large number of inputs in SI units. Note the return value is not standard SI. All parameters are required. This sizing model does not officially apply to liquid mixtures, slurries, non-Newtonian fluids, or liquid-solid conveyance systems. For details of the calculations, consult *[R208]*.

Parameters **rho** : float

Density of the liquid at the inlet [kg/m^3]

Psat : float

Saturation pressure of the fluid at inlet temperature [Pa]

Pc : float

Critical pressure of the fluid [Pa]

mu : float

Viscosity of the fluid [Pa*s]

P1 : float

Inlet pressure of the fluid before valves and reducers [Pa]

P2 : float

Outlet pressure of the fluid after valves and reducers [Pa]

Q : float

Volumetric flow rate of the fluid [m^3/s]

D1 : float

Diameter of the pipe before the valve [m]

D2 : float

Diameter of the pipe after the valve [m]

d : float

Diameter of the valve [m]

FL : float

Liquid pressure recovery factor of a control valve without attached fittings []

Fd : float

Valve style modifier []

Returns **C** : float

Kv flow coefficient [m^3/hr at a dP of 1 bar]

References

[R208]

Examples

From *[R208]*, matching example 1 for a globe, parabolic plug, flow-to-open valve.

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
... P1=680E3, P2=220E3, Q=0.1, D1=0.15, D2=0.15, d=0.15,
... FL=0.9, Fd=0.46)
164.9954763704956
```

From *[R208]*, matching example 2 for a ball, segmented ball, flow-to-open valve.

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
... P1=680E3, P2=220E3, Q=0.1, D1=0.1, D2=0.1, d=0.1,
... FL=0.6, Fd=0.98)
238.05817216710483
```

Modified example 1 with non-choked flow, with reducer and expander

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
...  P1=680E3, P2=220E3, Q=0.1, D1=0.1, D2=0.09, d=0.08, FL=0.9, Fd=0.46)
177.44417090966715
```

Modified example 2 with non-choked flow, with reducer and expander

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
... P1=680E3, P2=220E3, Q=0.1, D1=0.1, D2=0.1, d=0.95, FL=0.6, Fd=0.98)
230.1734424266345
```

Modified example 2 with laminar flow at 100x viscosity, 100th flow rate, and 1/10th diameters:

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-2,
... P1=680E3, P2=220E3, Q=0.001, D1=0.01, D2=0.01, d=0.01, FL=0.6, Fd=0.98)
3.0947562381723626
```

fluids.control_valve.**size_control_valve_g**(*T*, *MW*, *mu*, *gamma*, *Z*, *P1*, *P2*, *Q*, *D1*, *D2*, *d*, *FL*, *Fd*, *xT*)

Calculates flow coefficient of a control valve passing a gas according to IEC 60534. Uses a large number of inputs in SI units. Note the return value is not standard SI. All parameters are required. For details of the calculations, consult *[R209]*. Note the inlet gas flow conditions.

Parameters **T** : float

Temperature of the gas at the inlet [K]

MW : float

Molecular weight of the gas [g/mol]

mu : float

Viscosity of the fluid at inlet conditions [Pa*s]

gamma : float

Specific heat capacity ratio [-]

Z : float

Compressibility factor at inlet conditions, [-]

P1 : float

Inlet pressure of the gas before valves and reducers [Pa]

P2 : float

Outlet pressure of the gas after valves and reducers [Pa]

Q : float

Volumetric flow rate of the gas at *273.15 K* and 1 atm specifically [m^3/s]

D1 : float

Diameter of the pipe before the valve [m]

D2 : float

Diameter of the pipe after the valve [m]

d : float

Diameter of the valve [m]

FL : float

Liquid pressure recovery factor of a control valve without attached fittings []

Fd : float

Valve style modifier []

xT : float

Pressure difference ratio factor of a valve without fittings at choked flow [-]

Returns **C** : float

Kv flow coefficient [m^3/hr at a dP of 1 bar]

References

[R209]

Examples

From *[R209]*, matching example 3 for non-choked gas flow with attached fittings and a rotary, eccentric plug, flow-to-open control valve:

```
>>> size_control_valve_g(T=433., MW=44.01, mu=1.4665E-4, gamma=1.30,
... Z=0.988, P1=680E3, P2=310E3, Q=38/36., D1=0.08, D2=0.1, d=0.05,
... FL=0.85, Fd=0.42, xT=0.60)
72.58664545391052
```

From *[R209]*, roughly matching example 4 for a small flow trim sized tapered needle plug valve. Difference is 3% and explained by the difference in algorithms used.

```
>>> size_control_valve_g(T=320., MW=39.95, mu=5.625E-5, gamma=1.67, Z=1.0,
... P1=2.8E5, P2=1.3E5, Q=0.46/3600., D1=0.015, D2=0.015, d=0.015, FL=0.98,
... Fd=0.07, xT=0.8)
0.016498765335995726
```

fluids.core module

`fluids.core.`**`Reynolds`**(*V*, *D*, *rho=None*, *mu=None*, *nu=None*)

Calculates Reynolds number or *Re* for a fluid with the given properties for the specified velocity and diameter.

$$Re = D \cdot V \nu = \frac{\rho V D}{\mu}$$

Inputs either of any of the following sets:

- V, D, density *rho* and kinematic viscosity *mu*
- V, D, and dynamic viscosity *nu*

Parameters **D** : float

Diameter [m]

V : float

Velocity [m/s]

rho : float, optional

Density, [kg/m^3]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

Returns **Re** : float

Reynolds number []

Notes

$$Re = \frac{\text{Momentum}}{\text{Viscosity}}$$

An error is raised if none of the required input sets are provided.

References

[R210], *[R211]*

Examples

```
>>> Reynolds(2.5, 0.25, 1.1613, 1.9E-5)
38200.65789473684
>>> Reynolds(2.5, 0.25, nu=1.636e-05)
38202.93398533008
```

fluids.core.**Prandtl**(*Cp=None*, *k=None*, *mu=None*, *nu=None*, *rho=None*, *alpha=None*)

Calculates Prandtl number or *Pr* for a fluid with the given parameters.

$$Pr = \frac{C_p \mu}{k} = \frac{\nu}{\alpha} = \frac{C_p \rho \nu}{k}$$

Inputs can be any of the following sets:

- Heat capacity, dynamic viscosity, and thermal conductivity
- Thermal diffusivity and kinematic viscosity
- Heat capacity, kinematic viscosity, thermal conductivity, and density

Parameters **Cp** : float

Heat capacity, [J/kg/K]

k : float

Thermal conductivity, [W/m/K]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

rho : float

Density, [kg/m^3]

alpha : float

Thermal diffusivity, [m^2/s]

Returns **Pr** : float

Prandtl number []

Notes

$$Pr = \frac{\text{kinematic viscosity}}{\text{thermal diffusivity}} = \frac{\text{momentum diffusivity}}{\text{thermal diffusivity}}$$

An error is raised if none of the required input sets are provided.

References

[R212], *[R213]*, *[R214]*

Examples

```
>>> Prandtl(Cp=1637., k=0.010, mu=4.61E-6)
0.754657
>>> Prandtl(Cp=1637., k=0.010, nu=6.4E-7, rho=7.1)
0.7438528
>>> Prandtl(nu=6.3E-7, alpha=9E-7)
0.7000000000000001
```

fluids.core.**Grashof**(*L*, *beta*, *T1*, *T2=0*, *rho=None*, *mu=None*, *nu=None*, *g=9.80665*)

Calculates Grashof number or *Gr* for a fluid with the given properties, temperature difference, and characteristic length.

$$Gr = \frac{g\beta(T_s - T_\infty)L^3}{\nu^2} = \frac{g\beta(T_s - T_\infty)L^3\rho^2}{\mu^2}$$

Inputs either of any of the following sets:

- L, beta, T1 and T2, and density *rho* and kinematic viscosity *mu*
- L, beta, T1 and T2, and dynamic viscosity *nu*

Parameters **L** : float

Characteristic length [m]

beta : float

Volumetric thermal expansion coefficient [1/K]

T1 : float

Temperature 1, usually a film temperature [K]

T2 : float, optional

Temperature 2, usually a bulk temperature (or 0 if only a difference is provided to the function) [K]

rho : float, optional

Density, [kg/m^3]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

g : float, optional

Acceleration due to gravity, [m/s^2]

Returns **Gr** : float

Grashof number []

Notes

$$Gr = \frac{\text{Buoyancy forces}}{\text{Viscous forces}}$$

An error is raised if none of the required input sets are provided. Used in free convection problems only.

References

[R215], *[R216]*

Examples

Example 4 of *[R215]*, p. 1-21 (matches):

```
>>> Grashof(L=0.9144, beta=0.000933, T1=178.2, rho=1.1613, mu=1.9E-5)
4656936556.178915
>>> Grashof(L=0.9144, beta=0.000933, T1=378.2, T2=200, nu=1.636e-05)
4657491516.530312
```

fluids.core.**Nusselt**(*h*, *L*, *k*)

Calculates Nusselt number *Nu* for a heat transfer coefficient *h*, characteristic length *L*, and thermal conductivity *k*.

$$Nu = \frac{hL}{k}$$

Parameters **h** : float

Heat transfer coefficient, [W/m^2/K]

L : float

Characteristic length, no typical definition [m]

k : float

Thermal conductivity of fluid [W/m/K]

Returns **Nu** : float

Nusselt number, [-]

Notes

Do not confuse k, the thermal conductivity of the fluid, with that of within a solid object associated with!

$$Nu = \frac{\text{Convective heat transfer}}{\text{Conductive heat transfer}}$$

References

[R217], *[R218]*

Examples

```
>>> Nusselt(1000., 1.2, 300.)
4.0
>>> Nusselt(10000., .01, 4000.)
0.025
```

fluids.core.**Sherwood**(*K*, *L*, *D*)

Calculates Sherwood number *Sh* for a mass transfer coefficient *K*, characteristic length *L*, and diffusivity *D*.

$$Sh = \frac{KL}{D}$$

Parameters **K** : float

Mass transfer coefficient, [m/s]

L : float

Characteristic length, no typical definition [m]

D : float

Diffusivity of a species [m/s^2]

Returns **Sh** : float

Sherwood number, [-]

Notes

$$Sh = \frac{\text{Mass transfer by convection}}{\text{Mass transfer by diffusion}} = \frac{K}{D/L}$$

References

[R219]

Examples

```
>>> Sherwood(1000., 1.2, 300.)
4.0
```

fluids.core.**Rayleigh**(*Pr*, *Gr*)

Calculates Rayleigh number or *Ra* using Prandtl number *Pr* and Grashof number *Gr* for a fluid with the given properties, temperature difference, and characteristic length used to calculate *Gr* and *Pr*.

$$Ra = PrGr$$

Parameters **Pr** : float

Prandtl number []

Gr : float

Grashof number []

Returns **Ra** : float

Rayleigh number []

Notes

Used in free convection problems only.

References

[R220], *[R221]*

Examples

```
>>> Rayleigh(1.2, 4.6E9)
5520000000.0
```

fluids.core.**Schmidt**(*D*, *mu=None*, *nu=None*, *rho=None*)

Calculates Schmidt number or *Sc* for a fluid with the given parameters.

$$Sc = \frac{\mu}{D\rho} = \frac{\nu}{D}$$

Inputs can be any of the following sets:

- Diffusivity, dynamic viscosity, and density
- Diffusivity and kinematic viscosity

Parameters **D** : float

Diffusivity of a species, [m^2/s]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

rho : float, optional

Density, [kg/m^3]

Returns **Sc** : float

Schmidt number []

Notes

$$Sc = \frac{\text{kinematic viscosity}}{\text{molecular diffusivity}} = \frac{\text{viscous diffusivity}}{\text{species diffusivity}}$$

An error is raised if none of the required input sets are provided.

References

[R222], *[R223]*

Examples

```
>>> Schmidt(D=2E-6, mu=4.61E-6, rho=800)
0.00288125
>>> Schmidt(D=1E-9, nu=6E-7)
599.9999999999999
```

fluids.core.**Peclet_heat**(*V*, *L*, *rho=None*, *Cp=None*, *k=None*, *alpha=None*)

Calculates heat transfer Peclet number or *Pe* for a specified velocity *V*, characteristic length *L*, and specified properties for the given fluid.

$$Pe = \frac{VL\rho C_p}{k} = \frac{LV}{\alpha}$$

Inputs either of any of the following sets:

- V, L, density *rho*, heat capcity *Cp*, and thermal conductivity *k*
- V, L, and thermal diffusivity *alpha*

Parameters **V** : float

Velocity [m/s]

L : float

Characteristic length [m]

rho : float, optional

Density, [kg/m^3]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Returns **Pe** : float

Peclet number (heat) []

Notes

$$Pe = \frac{\text{Bulk heat transfer}}{\text{Conduction heat transfer}}$$

An error is raised if none of the required input sets are provided.

References

[R224], *[R225]*

Examples

```
>>> Peclet_heat(1.5, 2, 1000., 4000., 0.6)
20000000.0
>>> Peclet_heat(1.5, 2, alpha=1E-7)
30000000.0
```

fluids.core.**Peclet_mass**(*V*, *L*, *D*)

Calculates mass transfer Peclet number or *Pe* for a specified velocity *V*, characteristic length *L*, and diffusion coefficient *D*.

$$Pe = \frac{LV}{D}$$

Parameters **V** : float

Velocity [m/s]

L : float

Characteristic length [m]

D : float

Diffusivity of a species, [m^2/s]

Returns **Pe** : float

Peclet number (mass) []

Notes

$$Pe = \frac{\text{Advective transport rate}}{\text{Diffusive transport rate}}$$

References

[R226]

Examples

```
>>> Peclet_mass(1.5, 2, 1E-9)
3000000000.0
```

fluids.core.**Fourier_heat**(*t*, *L*, *rho=None*, *Cp=None*, *k=None*, *alpha=None*)

Calculates heat transfer Fourier number or *Fo* for a specified time *t*, characteristic length *L*, and specified properties for the given fluid.

$$Fo = \frac{kt}{C_p \rho L^2} = \frac{\alpha t}{L^2}$$

Inputs either of any of the following sets:

- t, L, density *rho*, heat capcity *Cp*, and thermal conductivity *k*
- t, L, and thermal diffusivity *alpha*

Parameters **t** : float

time [s]

L : float

Characteristic length [m]

rho : float, optional

Density, [kg/m^3]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Returns **Fo** : float

Fourier number (heat) []

Notes

$$Fo = \frac{\text{Heat conduction rate}}{\text{Rate of thermal energy storage in a solid}}$$

An error is raised if none of the required input sets are provided.

References

[R227], *[R228]*

Examples

```
>>> Fourier_heat(1.5, 2, 1000., 4000., 0.6)
5.625e-08
>>> Fourier_heat(1.5, 2, alpha=1E-7)
3.75e-08
```

fluids.core.**Fourier_mass**(*t*, *L*, *D*)

Calculates mass transfer Fourier number or *Fo* for a specified time *t*, characteristic length *L*, and diffusion coefficient *D*.

$$Fo = \frac{Dt}{L^2}$$

Parameters **t** : float

time [s]

L : float

Characteristic length [m]

D : float

Diffusivity of a species, [m^2/s]

Returns **Fo** : float

Fourier number (mass) []

Notes

$$Fo = \frac{\text{Diffusive transport rate}}{\text{Storage rate}}$$

References

[R229]

Examples

```
>>> Fourier_mass(1.5, 2, 1E-9)
3.7500000000000005e-10
```

fluids.core.**Graetz_heat**(*V*, *D*, *x*, *rho=None*, *Cp=None*, *k=None*, *alpha=None*)

Calculates Graetz number or *Gz* for a specified velocity *V*, diameter *D*, axial diatance *x*, and specified properties for the given fluid.

$$Gz = \frac{VD^2 \cdot C_p\rho}{x \cdot k} = \frac{VD^2}{x\alpha}$$

Inputs either of any of the following sets:

- V, D, x, density *rho*, heat capcity *Cp*, and thermal conductivity *k*
- V, D, x, and thermal diffusivity *alpha*

Parameters **V** : float

Velocity, [m/s]

D : float

Diameter [m]

x : float

Axial distance [m]

rho : float, optional

Density, [kg/m^3]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Returns **Gz** : float

Graetz number []

Notes

$$Gz = \frac{\text{Time for radial heat diffusion in a fluid by conduction}}{\text{Time taken by fluid to reach distance x}}$$

$$Gz = \frac{D}{x} RePr$$

An error is raised if none of the required input sets are provided.

References

[R230]

Examples

```
>>> Graetz_heat(1.5, 0.25, 5, 800., 2200., 0.6)
55000.0
>>> Graetz_heat(1.5, 0.25, 5, alpha=1E-7)
187500.0
```

`fluids.core.`**`Lewis`**(*D=None*, *alpha=None*, *Cp=None*, *k=None*, *rho=None*)

Calculates Lewis number or *Le* for a fluid with the given parameters.

$$Le = \frac{k}{\rho C_p D} = \frac{\alpha}{D}$$

Inputs can be either of the following sets:

•Diffusivity and Thermal diffusivity

•Diffusivity, heat capacity, thermal conductivity, and density

Parameters **D** : float

Diffusivity of a species, [m^2/s]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

rho : float, optional

Density, [kg/m^3]

Returns **Le** : float

Lewis number []

Notes

$$Le = \frac{\text{Thermal diffusivity}}{\text{Mass diffusivity}} = \frac{Sc}{Pr}$$

An error is raised if none of the required input sets are provided.

References

[R231], *[R232]*, *[R233]*

Examples

```
>>> Lewis(D=22.6E-6, alpha=19.1E-6)
0.8451327433628318
>>> Lewis(D=22.6E-6, rho=800., k=.2, Cp=2200)
0.00502815768302494
```

`fluids.core.`**`Weber`**(*V*, *L*, *rho*, *sigma*)

Calculates Weber number, *We*, for a fluid with the given density, surface tension, velocity, and geometric parameter (usually diameter of bubble).

$$We = \frac{V^2 L \rho}{\sigma}$$

Parameters **V** : float

Velocity of fluid, [m/s]

L : float

Characteristic length, typically bubble diameter [m]

rho : float

Density of fluid, [kg/m^3]

sigma : float

Surface tension, [N/m]

Returns **We** : float

Weber number []

Notes

Used in bubble calculations.

$$We = \frac{\text{inertial force}}{\text{surface tension force}}$$

References

[R234], *[R235]*, *[R236]*

Examples

```
>>> Weber(V=0.18, L=0.001, rho=900., sigma=0.01)
2.916
```

fluids.core.**Mach**(*V*, *c*)

Calculates Mach number or *Ma* for a fluid of velocity *V* with speed of sound *c*.

$$Ma = \frac{V}{c}$$

Parameters **V** : float

Velocity of fluid, [m/s]

c : float

Speed of sound in fluid, [m/s]

Returns **Ma** : float

Mach number []

Notes

Used in compressible flow calculations.

$$Ma = \frac{\text{fluid velocity}}{\text{sonic velocity}}$$

References

[R237], *[R238]*

Examples

```
>>> Mach(33., 330)
0.1
```

fluids.core.**Knudsen**(*path*, *L*)

Calculates Knudsen number or *Kn* for a fluid with mean free path *path* and for a characteristic length *L*.

$$Kn = \frac{\lambda}{L}$$

Parameters **path** : float

Mean free path between molecular collisions, [m]

L : float

Characteristic length, [m]

Returns **Kn** : float

Knudsen number []

Notes

Used in mass transfer calculations.

$$Kn = \frac{\text{Mean free path length}}{\text{Characteristic length}}$$

References

[R239], *[R240]*

Examples

```
>>> Knudsen(1e-10, .001)
1e-07
```

fluids.core.**Bond**(*rhol*, *rhog*, *sigma*, *L*)

Calculates Bond number, *Bo* also known as Eotvos number, for a fluid with the given liquid and gas densities, surface tension, and geometric parameter (usually length).

$$Bo = \frac{g(\rho_l - \rho_g)L^2}{\sigma}$$

Parameters **rhol** : float

Density of liquid, [kg/m^3]

rhog : float

Density of gas, [kg/m^3]

sigma : float

Surface tension, [N/m]

L : float

Characteristic length, [m]

Returns **Bo** : float

Bond number []

References

[R241]

Examples

```
>>> Bond(1000., 1.2, .0589, 2)
665187.2339558573
```

fluids.core.**Froude**(*V*, *L*, *g=9.80665*, *squared=False*)

Calculates Froude number *Fr* for velocity *V* and geometric length *L*. If desired, gravity can be specified as well. Normally the function returns the result of the equation below; Froude number is also often said to be defined as the square of the equation below.

$$Fr = \frac{V}{\sqrt{gL}}$$

Parameters **V** : float

Velocity of the particle or fluid, [m/s]

L : float

Characteristic length, no typical definition [m]

g : float, optional

Acceleration due to gravity, [m/s^2]

squared : bool, optional

Whether to return the squared form of Frounde number

Returns **Fr** : float

Froude number, [-]

Notes

Many alternate definitions including density ratios have been used.

$$Fr = \frac{\text{Inertial Force}}{\text{Gravity Force}}$$

References

[R242], *[R243]*

Examples

```
>>> Froude(1.83, L=2., g=1.63)
1.0135432593877318
>>> Froude(1.83, L=2., squared=True)
0.17074638128208924
```

fluids.core.**Strouhal**(*f*, *L*, *V*)

Calculates Strouhal number *St* for a characteristic frequency *f*, characteristic length *L*, and velocity *V*.

$$St = \frac{fL}{V}$$

Parameters **f** : float

Characteristic frequency, usually that of vortex shedding, [Hz]

L : float

Characteristic length, [m]

V : float

Velocity of the fluid, [m/s]

Returns **St** : float

Strouhal number, [-]

Notes

Sometimes abbreviated to S or Sr.

$$St = \frac{\text{Characteristif flow time}}{\text{Period of oscillation}}$$

References

[R244], *[R245]*

Examples

```
>>> Strouhal(8, 2., 4.)
4.0
```

fluids.core.**Biot**(*h*, *L*, *k*)

Calculates Biot number *Br* for heat transfer coefficient *h*, geometric length *L*, and thermal conductivity *k*.

$$Bi = \frac{hL}{k}$$

Parameters **h** : float

Heat transfer coefficient, [W/m^2/K]

L : float

Characteristic length, no typical definition [m]

k : float

Thermal conductivity, within the object [W/m/K]

Returns **Bi** : float

Biot number, [-]

Notes

Do not confuse k, the thermal conductivity within the object, with that of the medium h is calculated with!

$$Bi = \frac{\text{Surface thermal resistance}}{\text{Internal thermal resistance}}$$

References

[R246], *[R247]*

Examples

```
>>> Biot(1000., 1.2, 300.)
4.0
>>> Biot(10000., .01, 4000.)
0.025
```

`fluids.core.`**`Stanton`**(*h*, *V*, *rho*, *Cp*)

Calculates Stanton number or *St* for a specified heat transfer coefficient *h*, velocity *V*, density *rho*, and heat capacity *Cp*.

$$St = \frac{h}{V\rho Cp}$$

Parameters **h** : float

Heat transfer coefficient, [W/m^2/K]

V : float

Velocity, [m/s]

rho : float

Density, [kg/m^3]

Cp : float

Heat capacity, [J/kg/K]

Returns **St** : float

Stanton number []

Notes

$$St = \frac{\text{Heat transfer coefficient}}{\text{Thermal capacity}}$$

References

[R248], *[R248]*

Examples

```
>>> Stanton(5000, 5, 800, 2000.)
0.000625
```

fluids.core.**Euler**(*dP*, *rho*, *V*)

Calculates Euler number or *Eu* for a fluid of velocity *V* and density *rho* experiencing a pressure drop *dP*.

$$Eu = \frac{\Delta P}{\rho V^2}$$

Parameters **dP** : float

Pressure drop experience by the fluid, [Pa]

rho : float

Density of the fluid, [kg/m^3]

V : float

Velocity of fluid, [m/s]

Returns **Eu** : float

Euler number []

Notes

Used in pressure drop calculations. Rarely, this number is divided by two. Named after Leonhard Euler applied calculus to fluid dynamics.

$$Eu = \frac{\text{Pressure drop}}{2 \cdot \text{velocity head}}$$

References

[R250], *[R251]*

Examples

```
>>> Euler(1E5, 1000., 4)
6.25
```

fluids.core.**Cavitation**(*P*, *Psat*, *rho*, *V*)

Calculates Cavitation number or *Ca* for a fluid of velocity *V* with a pressure *P*, vapor pressure *Psat*, and density *rho*.

$$Ca = \sigma_c = \sigma = \frac{P - P_{sat}}{\frac{1}{2}\rho V^2}$$

Parameters **P** : float

Internal pressure of the fluid, [Pa]

Psat : float

Vapor pressure of the fluid, [Pa]

rho : float

Density of the fluid, [kg/m^3]

V : float

Velocity of fluid, [m/s]

Returns **Ca** : float

Cavitation number []

Notes

Used in determining if a flow through a restriction will cavitate. Sometimes, the multiplication by 2 will be omitted;

$$Ca = \frac{\text{Pressure - Vapor pressure}}{\text{Inertial pressure}}$$

References

[R252], *[R253]*

Examples

```
>>> Cavitation(2E5, 1E4, 1000, 10)
3.8
```

fluids.core.**Eckert**(*V*, *Cp*, *dT*)

Calculates Eckert number or *Ec* for a fluid of velocity *V* with a heat capacity *Cp*, between two temperature given as *dT*.

$$Ec = \frac{V^2}{C_p \Delta T}$$

Parameters **V** : float

Velocity of fluid, [m/s]

Cp : float

Heat capacity of the fluid, [J/kg/K]

dT : float

Temperature difference, [K]

Returns **Ec** : float

Eckert number []

Notes

Used in certain heat transfer calculations. Fairly rare.

$$Ec = \frac{\text{Kinetic energy}}{\text{Enthalpy difference}}$$

References

[R254]

Examples

```
>>> Eckert(10, 2000., 25.)
0.002
```

fluids.core.**Jakob**(*Cp*, *Hvap*, *Te*)

Calculates Jakob number or *Ja* for a boiling fluid with sensible heat capacity *Cp*, enthalpy of vaporization *Hvap*, and boiling at *Te* degrees above its saturation boiling point.

$$Ja = \frac{C_P \Delta T_e}{\Delta H_{vap}}$$

Parameters **Cp** : float

Heat capacity of the fluid, [J/kg/K]

Hvap : float

Enthalpy of vaporization of the fluid at its saturation temperature [J/kg]

Te : float

Temperature difference above the fluid's saturation boiling temperature, [K]

Returns **Ja** : float

Jakob number []

Notes

Used in boiling heat transfer analysis. Fairly rare.

$$Ja = \frac{\Delta\text{Sensible heat}}{\Delta\text{Latent heat}}$$

References

[R255], *[R256]*

Examples

```
>>> Jakob(4000., 2E6, 10.)
0.02
```

fluids.core.**Power_number**(*P*, *L*, *N*, *rho*)

Calculates power number, *Po*, for an agitator applying a specified power *P* with a characteristic length *L*, rotationa speed *N*, to a fluid with a specified density *rho*.

$$Po = \frac{P}{\rho N^3 D^5}$$

Parameters **P** : float

Power applied, [W]

L : float

Characteristic length, typically agitator diameter [m]

N : float

Speed [revolutions/second]

rho : float

Density of fluid, [kg/m^3]

Returns **Po** : float

Power number []

Notes

Used in mixing calculations.

$$Po = \frac{\text{Power}}{\text{Rotational inertia}}$$

References

[R257], *[R258]*

Examples

```
>>> Power_number(P=180, L=0.01, N=2.5, rho=800.)
144000000.0
```

fluids.core.**Drag**(*F*, *A*, *V*, *rho*)

Calculates drag coefficient *Cd* for a given drag force *F*, projected area *A*, characteristic velocity *V*, and density *rho*.

$$C_D = \frac{F_d}{A \cdot \frac{1}{2}\rho V^2}$$

Parameters **F** : float

Drag force, [N]

A : float

Projected area, [m^2]

V : float

Characteristic velocity, [m/s]

rho : float

Density, [kg/m^3]

Returns **Cd** : float

Drag coefficient, [-]

Notes

Used in flow around objects, or objects flowing within a fluid.

$$C_D = \frac{\text{Drag forces}}{\text{Projected area} \cdot \text{Velocity head}}$$

References

[R259], *[R260]*

Examples

```
>>> Drag(1000, 0.0001, 5, 2000)
400.0
```

fluids.core.**Capillary**(*V*, *mu*, *sigma*)

Calculates Capillary number *Ca* for a characteristic velocity *V*, viscosity *mu*, and surface tension *sigma*.

$$Ca = \frac{V\mu}{\sigma}$$

Parameters **V** : float

Characteristic velocity, [m/s]

mu : float

Dynamic viscosity, [Pa*s]

sigma : float

Surface tension, [N/m]

Returns **Ca** : float

Capillary number, [-]

Notes

Used in porous media calculations and film flow calculations. Surface tension may gas-liquid, or liquid-liquid.

$$Ca = \frac{\text{Viscous forces}}{\text{Surface forces}}$$

References

[R261], *[R262]*

Examples

```
>>> Capillary(1.2, 0.01, .1)
0.12
```

fluids.core.**Archimedes**(*L*, *rhof*, *rhop*, *mu*, *g=9.80665*)

Calculates Archimedes number, *Ar*, for a fluid and particle with the given densities, characteristic length, viscosity, and gravity (usually diameter of particle).

$$Ar = \frac{L^3 \rho_f (\rho_p - \rho_f) g}{\mu^2}$$

Parameters **L** : float

Characteristic length, typically particle diameter [m]

rhof : float

Density of fluid, [kg/m^3]

rhop : float

Density of particle, [kg/m^3]

mu : float

Viscosity of fluid, [N/m]

g : float, optional

Acceleration due to gravity, [m/s^2]

Returns **Ar** : float

Archimedes number []

Notes

Used in fluid-particle interaction calculations.

$$Ar = \frac{\text{Gravitational force}}{\text{Viscous force}}$$

References

[R263], *[R264]*

Examples

```
>>> Archimedes(0.002, 0.2804, 2699.37, 4E-5)
37109.575890227665
```

fluids.core.**Ohnesorge**(*L*, *rho*, *mu*, *sigma*)

Calculates Ohnesorge number, *Oh*, for a fluid with the given characteristic length, density, viscosity, and surface tension.

$$\text{Oh} = \frac{\mu}{\sqrt{\rho \sigma L}}$$

Parameters **L** : float

Characteristic length [m]

rho : float

Density of fluid, [kg/m^3]

mu : float

Viscosity of fluid, [Pa*s]

sigma : float

Surface tension, [N/m]

Returns **Oh** : float

Ohnesorge number []

Notes

Often used in spray calculations. Sometimes given the symbol Z.

$$Oh = \frac{\sqrt{\text{We}}}{\text{Re}} = \frac{\text{viscous forces}}{\sqrt{\text{Inertia} \cdot \text{Surface tension}}}$$

References

[R265]

Examples

```
>>> Ohnesorge(1E-5, 2000., 1E-4, 1E-1)
0.00223606797749979
```

fluids.core.**thermal_diffusivity**(*k*, *rho*, *Cp*)

Calculates thermal diffusivity or *alpha* for a fluid with the given parameters.

$$\alpha = \frac{k}{\rho Cp}$$

Parameters **k** : float

Thermal conductivity, [W/m/K]

Cp : float

Heat capacity, [J/kg/K]

rho : float

Density, [kg/m^3]

Returns **alpha** : float

Thermal diffusivity, [m^2/s]

References

[R266]

Examples

```
>>> thermal_diffusivity(0.02, 1., 1000.)
2e-05
```

fluids.core.**c_ideal_gas**(*T*, *k*, *MW*)

Calculates speed of sound *c* in an ideal gas at tempereture T.

$$c = \sqrt{kR_{specific}T}$$

Parameters **T** : float

Temperature of fluid, [K]

k : float

Isentropic exponent of fluid, [-]

MW : float

Molecular weight of fluid, [g/mol]

Returns **c** : float

Speed of sound in fluid, [m/s]

Notes

Used in compressible flow calculations. Note that the gas constant used is the specific gas constant:

$$R_{specific} = R\frac{1000}{MW}$$

References

[R267], *[R268]*

Examples

```
>>> c_ideal_gas(1.4, 303., 28.96)
348.9820361755092
```

fluids.core.**relative_roughness**(*D*, *roughness=1.52e-06*)

Calculates relative roughness *eD* using a diameter and the roughness of the material of the wall. Default roughness is that of steel.

$$eD = \frac{\epsilon}{D}$$

Parameters **D** : float

Diameter of pipe, [m]

roughness : float, optional

Roughness of pipe wall [m]

Returns **eD** : float

Relative Roughness, [-]

References

[R269], *[R270]*

Examples

```
>>> relative_roughness(0.0254)
5.9842519685039374e-05
>>> relative_roughness(0.5, 1E-4)
0.0002
```

fluids.core.**nu_mu_converter**(*rho*, *nu=0*, *mu=0*)

Specify density and either Kinematic viscosity or Dynamic Viscosity by name for the result to be converted to the other.

```
>>> nu_mu_converter(998.1,nu=1.01E-6)
0.001008081
```

fluids.core.**gravity**(*latitude*, *height*)

Calculates local acceleration due to gravity g according to *[R271]*. Uses latitude and height to calculate g.

$$g = 9.780356(1 + 0.0052885\sin^2\phi - 0.0000059^2 2\phi) - 3.086\times 10^{-6}H$$

Parameters **latitude** : float

Degrees, [degrees]

height : float

Height above earth's surface [m]

Returns **g** : float

Acceleration due to gravity, [m/s^2]

Notes

Better models, such as EGM2008 exist.

References

[R271]

Examples

```
>>> gravity(55, 1E4)
9.784151976863571
```

fluids.core.**K_from_f**(*f*, *L*, *D*)

Calculates loss coefficient, K, for a given section of pipe at a specified friction factor.

$$K = fL/D$$

Parameters **f** : float

friction factor of pipe, []

L : float

Length of pipe, [m]

D : float

Inner diameter of pipe, [m]

Returns **K** : float

Loss coefficient, []

Notes

For fittings with a specified L/D ratio, use D = 1 and set L to specified L/D ratio.

Examples

```
>>> K_from_f(f=0.018, L=100., D=.3)
6.0
```

fluids.core.**K_from_L_equiv**(*L_D*, *f=0.015*)

Calculates loss coefficient, for a given equivalent length (L/D).

$$K = f\frac{L}{D}$$

Parameters **L_D** : float

Length over diameter, []

f : float, optional

Darcy friction factor, [-]

Returns **K** : float

Loss coefficient, []

Notes

Almost identical to *K_from_f*, but with a default friction factor for fully turbulent flow in steel pipes.

Examples

```
>>> K_from_L_equiv(240.)
3.5999999999999996
```

fluids.core.**dP_from_K**(*K*, *rho*, *V*)

Calculates pressure drop, for a given loss coefficient, at a specified density and velocity.

$$dP = 0.5 K \rho V^2$$

Parameters **K** : float

Loss coefficient, []

rho : float

Density of fluid, [kg/m^3]

V : float

Velocity of fluid in pipe, [m/s]

Returns **dP** : float

Pressure drop, [Pa]

Notes

Loss ciefficient *K* is usually the sum of several factors, including the friction factor.

Examples

```
>>> dP_from_K(K=10, rho=1000, V=3)
45000.0
```

fluids.core.**head_from_K**(*K*, *V*)

Calculates head loss, for a given loss coefficient, at a specified velocity.

$$\text{head} = \frac{KV^2}{2g}$$

Parameters **K** : float

Loss coefficient, []

V : float

Velocity of fluid in pipe, [m/s]

Returns **head** : float

Head loss, [m]

Notes

Loss ciefficient *K* is usually the sum of several factors, including the friction factor.

Examples

```
>>> head_from_K(K=10, V=1.5)
1.1471807396001694
```

fluids.core.**head_from_P**(*P*, *rho*)

Calculates head for a fluid of specified density at specified pressure.

$$\text{head} = \frac{P}{\rho g}$$

Parameters **P** : float

Pressure fluid in pipe, [Pa]

rho : float

Density of fluid, [kg/m^3]

Returns **head** : float

Head, [m]

Notes

By definition. Head varies with location, inversely propertional to the increase in gravitational constant.

Examples

```
>>> head_from_P(P=1E5, rho=1000.)
10.197162129779283
```

fluids.core.**P_from_head**(*head*, *rho*)

Calculates head for a fluid of specified density at specified pressure.

$$P = \rho g \cdot \text{head}$$

Parameters **head** : float

Head, [m]

rho : float

Density of fluid, [kg/m^3]

Returns **P** : float

Pressure fluid in pipe, [Pa]

Examples

```
>>> P_from_head(head=5., rho=800.)
39226.6
```

fluids.filters module

`fluids.filters.`**`round_edge_screen`**(*alpha*, *Re*, *angle=0*)

Returns the loss coefficient for a round edged wire screen or bar screen, as shown in *[R272]*. Angle of inclination may be specified as well.

Parameters **alpha** : float

Fraction of screen open to flow [-]

Re : float

Reynolds number of flow through screen with D = space between rods, []

angle : float, optional

Angle of inclination, with 0 being straight and 90 being parallel to flow [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

Linear interpolation between a table of values. Re should be > 20. alpha should be between 0.05 and 0.8. Angle may be no more than 85 degress.

References

[R272]

Examples

```
>>> round_edge_screen(0.5, 100)
2.0999999999999996
>>> round_edge_screen(0.5, 100, 45)
1.05
>>> round_edge_screen(0.5, 100, 85)
0.18899999999999997
```

`fluids.filters.`**`round_edge_open_mesh`**(*alpha*, *subtype='diamond pattern wire'*, *angle=0*)

Returns the loss coefficient for a round edged open net/screen made of one of the following patterns, according to *[R273]*:

'round bar screen':

$$K = 0.95(1-\alpha) + 0.2(1-\alpha)^2$$

'diamond pattern wire':

$$K = 0.67(1-\alpha) + 1.3(1-\alpha)^2$$

'knotted net':

$$K = 0.70(1-\alpha) + 4.9(1-\alpha)^2$$

'knotless net':

$$K = 0.72(1-\alpha) + 2.1(1-\alpha)^2$$

Parameters **alpha** : float

Fraction of net/screen open to flow [-]

subtype : str

One of 'round bar screen', 'diamond pattern wire', 'knotted net' or 'knotless net'.

angle : float, optional

Angle of inclination, with 0 being straight and 90 being parallel to flow [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

alpha should be between 0.85 and 1 for these correlations. Flow should be turbulent, with Re > 500. Angle may be no more than 85 degress.

References

[R273]

Examples

```
>>> [round_edge_open_mesh(0.88, i) for i in ['round bar screen',
... 'diamond pattern wire', 'knotted net', 'knotless net']]
[0.11687999999999998, 0.09912, 0.15455999999999998, 0.11664]
>>> round_edge_open_mesh(0.96, angle=33.)
0.02031327712601458
```

`fluids.filters.`**`square_edge_screen`**(*alpha*)

Returns the loss coefficient for a square wire screen or square bar screen or perforated plate with squared edges, as shown in *[R274]*.

Parameters **alpha** : float

Fraction of screen open to flow [-]

Returns **K** : float

Loss coefficient [-]

Notes

Linear interpolation between a table of values.

References

[R274]

Examples

```
>>> square_edge_screen(0.99)
0.008000000000000007
```

fluids.filters.**square_edge_grill**(*alpha=None*, *l=None*, *Dh=None*, *fd=None*)

Returns the loss coefficient for a square grill or square bar screen or perforated plate with squared edges of thickness l, as shown in *[R275]*.

for Dh < l < 50D

$$K = \frac{0.5(1-\alpha) + (1-\alpha^2)}{\alpha^2}$$

else:

$$K = \frac{0.5(1-\alpha) + (1-\alpha^2) + fl/D}{\alpha^2}$$

Parameters **alpha** : float

Fraction of grill open to flow [-]

l : float

Thickness of the grill or plate [m]

Dh : float

Hydraulic diameter of gap in grill, [m]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

If l, Dh, or fd is not provided, the first expression is used instead. The alteration of the expression to include friction factor is there if the grill is long enough to have considerable friction along the surface of the grill.

References

[R275]

Examples

```
>>> square_edge_grill(.45)
5.296296296296296
>>> square_edge_grill(.45, l=.15, Dh=.002, fd=.0185)
12.148148148148147
```

fluids.filters.**round_edge_grill**(*alpha=None*, *l=None*, *Dh=None*, *fd=None*)

Returns the loss coefficient for a rounded square grill or square bar screen or perforated plate with rounded edges of thickness l, as shown in *[R276]*.

for Dh < l < 50D

$$K = lookup(alpha)$$

else:

$$K = lookup(alpha) + \frac{fl}{\alpha^2 D}$$

Parameters **alpha** : float

Fraction of grill open to flow [-]

l : float, optional

Thickness of the grill or plate [m]

Dh : float, optional

Hydraulic diameter of gap in grill, [m]

fd : float, optional

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

If l, Dh, or fd is not provided, the first expression is used instead. The alteration of the expression to include friction factor is there if the grill is long enough to have considerable friction along the surface of the grill. alpha must be between 0.3 and 0.7.

References

[R276]

Examples

```
>>> round_edge_grill(.45)
0.8
>>> round_edge_grill(.45, l=.15, Dh=.002, fd=.0185)
2.1875
```

fluids.fittings module

`fluids.fittings.`**`contraction_sharp`**(*Di1*, *Di2*)

Returns loss coefficient for any sharp edged pipe contraction as shown in *[R277]*.

$$K = 0.0696(1-\beta^5)\lambda^2 + (\lambda-1)^2$$
$$\lambda = 1 + 0.622(1 - 0.215\beta^2 - 0.785\beta^5)$$
$$\beta = d_2/d_1$$

Parameters **Di1** : float

Inside diameter of original pipe, [m]

Di2 : float

Inside diameter of following pipe, [m]

Returns **K** : float

Loss coefficient [-]

Notes

A value of 0.506 or simply 0.5 is often used.

References

[R277]

Examples

```
>>> contraction_sharp(Di1=1, Di2=0.4)
0.5301269161591805
```

`fluids.fittings.`**`contraction_round`**(*Di1*, *Di2*, *rc*)

Returns loss coefficient for any round edged pipe contraction as shown in *[R278]*.

$$K = 0.0696\left(1 - 0.569\frac{r}{d_2}\right)\left(1 - \sqrt{\frac{r}{d_2}}\beta\right)(1-\beta^5)\lambda^2 + (\lambda-1)^2$$
$$\lambda = 1 + 0.622\left(1 - 0.30\sqrt{\frac{r}{d_2}} - 0.70\frac{r}{d_2}\right)^4(1 - 0.215\beta^2 - 0.785\beta^5)$$
$$\beta = d_2/d_1$$

Parameters **Di1** : float

Inside diameter of original pipe, [m]

Di2 : float

Inside diameter of following pipe, [m]

rc : float

Radius of curvatuce of the contraction, [m]

Returns **K** : float

Loss coefficient [-]

Notes

Rounding radius larger than 0.14Di2 prevents flow separation from the wall. Further increase in rounding radius continues to reduce loss coefficient.

References

[R278]

Examples

```
>>> contraction_round(Di1=1, Di2=0.4, rc=0.04)
0.1783332490866574
```

fluids.fittings.**contraction_conical**(*Di1*, *Di2*, *l=None*, *angle=None*, *fd=None*)

Returns loss coefficient for any conical pipe contraction as shown in *[R279]*.

$$K = 0.0696[1 + C_B(\sin(\alpha/2) - 1)](1 - \beta^5)\lambda^2 + (\lambda - 1)^2$$
$$\lambda = 1 + 0.622(\alpha/180)^{0.8}(1 - 0.215\beta^2 - 0.785\beta^5)$$
$$\beta = d_2/d_1$$

Parameters **Di1** : float

Inside diameter of original pipe, [m]

Di2 : float

Inside diameter of following pipe, [m]

l : float

Length of the contraction, optional [m]

angle : float

Angle of contraction, optional [degrees]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

Cheap and has substantial impact on pressure drop.

References

[R279]

Examples

```
>>> contraction_conical(Di1=0.1, Di2=0.04, l=0.04, fd=0.0185)
0.15779041548350314
```

fluids.fittings.**contraction_beveled**(*Di1*, *Di2*, *l=None*, *angle=None*)

Returns loss coefficient for any sharp beveled pipe contraction as shown in *[R280]*.

$$K = 0.0696[1 + C_B(\sin(\alpha/2) - 1)](1 - \beta^5)\lambda^2 + (\lambda - 1)^2$$

$$\lambda = 1 + 0.622\left[1 + C_B\left(\left(\frac{\alpha}{180}\right)^{0.8} - 1\right)\right](1 - 0.215\beta^2 - 0.785\beta^5)$$

$$C_B = \frac{l}{d_2}\frac{2\beta\tan(\alpha/2)}{1-\beta}$$

$$\beta = d_2/d_1$$

Parameters **Di1** : float

Inside diameter of original pipe, [m]

Di2 : float

Inside diameter of following pipe, [m]

l : float

Length of the bevel along the pipe axis ,[m]

angle : float

Angle of bevel, [degrees]

Returns **K** : float

Loss coefficient [-]

References

[R280]

Examples

```
>>> contraction_beveled(Di1=0.5, Di2=0.1, l=.7*.1, angle=120)
0.40946469413070485
```

fluids.fittings.**diffuser_sharp**(*Di1*, *Di2*)

Returns loss coefficient for any sudded pipe diameter expansion as shown in *[R281]* and in other sources.

$$K_1 = (1 - \beta^2)^2$$

Parameters **Di1** : float

Inside diameter of original pipe (smaller), [m]

Di2 : float

Inside diameter of following pipe (larger), [m]

Returns **K** : float

Loss coefficient [-]

Notes

Highly accurate.

References

[R281]

Examples

```
>>> diffuser_sharp(Di1=.5, Di2=1)
0.5625
```

fluids.fittings.**diffuser_conical**(*Di1*, *Di2*, *l=None*, *angle=None*, *fd=None*)

Returns loss coefficient for any conical pipe expansion as shown in *[R282]*. Five different formulas are used, depending on the angle and the ratio of diameters.

For 0 to 20 degrees, all aspect ratios:

$$K_1 = 8.30[\tan(\alpha/2)]^{1.75}(1 - \beta^2)^2 + \frac{f(1 - \beta^4)}{8\sin(\alpha/2)}$$

For 20 to 60 degrees, beta < 0.5:

$$K_1 = \left\{1.366\sin\left[\frac{2\pi(\alpha - 15^\circ)}{180}\right]^{0.5} - 0.170 - 3.28(0.0625 - \beta^4)\sqrt{\frac{\alpha - 20^\circ}{40^\circ}}\right\}(1 - \beta^2)^2 + \frac{f(1 - \beta^4)}{8\sin(\alpha/2)}$$

For 20 to 60 degrees, beta >= 0.5:

$$K_1 = \left\{1.366\sin\left[\frac{2\pi(\alpha - 15^\circ)}{180}\right]^{0.5} - 0.170\right\}(1 - \beta^2)^2 + \frac{f(1 - \beta^4)}{8\sin(\alpha/2)}$$

For 60 to 180 degrees, beta < 0.5:

$$K_1 = \left[1.205 - 3.28(0.0625 - \beta^4) - 12.8\beta^6\sqrt{\frac{\alpha - 60^\circ}{120^\circ}}\right](1 - \beta^2)^2$$

For 60 to 180 degrees, beta >= 0.5:

$$K_1 = \left[1.205 - 0.20\sqrt{\frac{\alpha - 60^\circ}{120^\circ}}\right](1 - \beta^2)^2$$

Parameters **Di1** : float

Inside diameter of original pipe (smaller), [m]

Di2 : float

Inside diameter of following pipe (larger), [m]

l : float

Length of the contraction along the pipe axis, optional[m]

angle : float

Angle of contraction, [degrees]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

For angles above 60 degrees, friction factor is not used.

References

[R282]

Examples

```
>>> diffuser_conical(Di1=.1**0.5, Di2=1, angle=10., fd=0.020)
0.12301652230915454
>>> diffuser_conical(Di1=1/3., Di2=1, angle=50, fd=0.03) # 2
0.8081340270019336
>>> diffuser_conical(Di1=2/3., Di2=1, angle=40, fd=0.03) # 3
0.32533470783539786
>>> diffuser_conical(Di1=1/3., Di2=1, angle=120, fd=0.0185) # #4
0.812308728765127
>>> diffuser_conical(Di1=2/3., Di2=1, angle=120, fd=0.0185) # Last
0.3282650135070033
```

fluids.fittings.**diffuser_conical_staged**(*Di1*, *Di2*, *DEs*, *ls*, *fd=None*)

Returns loss coefficient for any series of staged conical pipe expansions as shown in *[R283]*. Five different formulas are used, depending on the angle and the ratio of diameters. This function calls diffuser_conical.

Parameters **Di1** : float

Inside diameter of original pipe (smaller), [m]

Di2 : float

Inside diameter of following pipe (larger), [m]

DEs : array

Diameters of intermediate sections, [m]

ls : array

Lengths of the various sections, [m]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

Only lengths of sections currently allowed. This could be changed to understand angles also.

Formula doesn't make much sense, as observed by the example comparing a series of conical sections. Use only for small numbers of segments of highly differing angles.

References

[R283]

Examples

```
>>> diffuser_conical_staged(Di1=1., Di2=10., DEs=[2,3,4,5,6,7,8,9], ls=[1,1,1,1,1,1,1,1,1], fd=0
1.7681854713484308
>>> diffuser_conical(Di1=1., Di2=10.,l=9, fd=0.01)
0.973137914861591
```

`fluids.fittings.`**`diffuser_curved`**(*Di1*, *Di2*, *l*)

Returns loss coefficient for any curved wall pipe expansion as shown in *[R284]*.

$$K_1 = \phi(1.43 - 1.3\beta^2)(1-\beta^2)^2$$

$$\phi = 1.01 - 0.624\frac{l}{d_1} + 0.30\left(\frac{l}{d_1}\right)^2 - 0.074\left(\frac{l}{d_1}\right)^3 + 0.0070\left(\frac{l}{d_1}\right)^4$$

Parameters **Di1** : float

Inside diameter of original pipe (smaller), [m]

Di2 : float

Inside diameter of following pipe (larger), [m]

l : float

Length of the curve along the pipe axis, [m]

Returns **K** : float

Loss coefficient [-]

Notes

Beta^2 should be between 0.1 and 0.9. A small mismatch between tabulated values of this function in table 11.3 is observed with the equation presented.

References

[R284]

Examples

```
>>> diffuser_curved(Di1=.25**0.5, Di2=1., l=2.)
0.2299781250000002
```

fluids.fittings.**diffuser_pipe_reducer**(*Di1*, *Di2*, *l*, *fd1*, *fd2=None*)

Returns loss coefficient for any pipe reducer pipe expansion as shown in [1]. This is an approximate formula.

$$K_f = f_1\frac{0.20l}{d_1} + \frac{f_1(1-\beta)}{8\sin(\alpha/2)} + f_2\frac{0.20l}{d_2}\beta^4$$

$$\alpha = 2\tan^{-1}\left(\frac{d_1-d_2}{1.20l}\right)$$

Parameters **Di1** : float

Inside diameter of original pipe (smaller), [m]

Di2 : float

Inside diameter of following pipe (larger), [m]

l : float

Length of the pipe reducer along the pipe axis, [m]

fd1 : float

Darcy friction factor at inlet diameter [-]

fd2 : float

Darcy friction factor at outlet diameter, optional [-]

Returns **K** : float

Loss coefficient [-]

Notes

Industry lack of standardization prevents better formulas from being developed. Add 15% if the reducer is eccentric. Friction factor at outlet will be assumed the same as at inlet if not specified.

Doubt about the validity of this equation is raised.

References

[R285]

Examples

```
>>> diffuser_pipe_reducer(Di1=.5, Di2=.75, l=1.5, fd1=0.07)
0.06873244301714816
```

fluids.fittings.**entrance_sharp**()

Returns loss coefficient for a sharp entrance to a pipe as shown in *[R286]*.

$$K = 0.57$$

Returns **K** : float

Loss coefficient [-]

Notes

Other values used have been 0.5.

References

[R286]

Examples

```
>>> entrance_sharp()
0.57
```

fluids.fittings.**entrance_distance**(*d=None*, *t=None*, *l=None*)

Returns loss coefficient for a sharp entrance to a pipe at a distance from the wall of a reservoir, as shown in *[R287]*.

$$K = 1.12 - 22\frac{t}{d} + 216\left(\frac{t}{d}\right)^2 + 80\left(\frac{t}{d}\right)^3$$

Parameters **Di** : float

Inside diameter of pipe, [m]

t : float

Thickness of pipe wall, [m]

l : float, optional

Length of pipe extending from the wall, [m]

Returns **K** : float

Loss coefficient [-]

Notes

Requires that l/d be >= 0.5. Requires that t/d <= 0.05.

References

[R287]

Examples

```
>>> entrance_distance(d=0.1, t=0.0005)
1.0154100000000004
```

fluids.fittings.**entrance_angled**(*angle*)

Returns loss coefficient for a sharp, angled entrance to a pipe flush with the wall of a reservoir, as shown in *[R288]*.

$$K = 0.57 + 0.30\cos(\theta) + 0.20\cos(\theta)^2$$

Parameters **angle** : float

Angle of inclination, [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

Not reliable for angles under 20 degrees. Loss coefficient is the same for a upward or downward angle.

References

[R288]

Examples

```
>>> entrance_angled(30)
0.9798076211353316
```

fluids.fittings.**entrance_rounded**(*Di*, *rc*)

Returns loss coefficient for a rounded entrance to a pipe flush with the wall of a reservoir, as shown in *[R289]*.

$$K = 0.0696\left(1 - 0.569\frac{r}{d}\right)\lambda^2 + (\lambda - 1)^2$$

$$\lambda = 1 + 0.622\left(1 - 0.30\sqrt{\frac{r}{d}} - 0.70\frac{r}{d}\right)^4$$

Parameters **Di** : float

Inside diameter of pipe, [m]

rd : float

Radius of curvatuce of the entrance, [m]

Returns **K** : float

Loss coefficient [-]

Notes

Applies for r/D < 1. For generously rounded entrances (r/D ~= 1): K = 0.03

References

[R289]

Examples

```
>>> entrance_rounded(Di=0.1, rc=0.0235)
0.09839534618360923
```

fluids.fittings.**entrance_beveled**(*Di*, *l*, *angle*)

Returns loss coefficient for a beveled entrance to a pipe flush with the wall of a reservoir, as shown in *[R290]*.

$$K = 0.0696\left(1 - C_b\frac{l}{d}\right)\lambda^2 + (\lambda - 1)^2$$

$$\lambda = 1 + 0.622\left[1 - 1.5C_b\left(\frac{l}{d}\right)^{\frac{1-(l/d)^{1/4}}{2}}\right]$$

$$C_b = \left(1 - \frac{\theta}{90}\right)\left(\frac{\theta}{90}\right)^{\frac{1}{l+l/d}}$$

Parameters **Di** : float

Inside diameter of pipe, [m]

l : float

Length of bevel, [m]

angle : float

Angle of bevel, [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

A cheap way of getting a lower pressure drop. Little credible data is available.

References

[R290]

Examples

```
>>> entrance_beveled(Di=0.1, l=0.003, angle=45)
0.45086864221916984
```

fluids.fittings.**exit_normal**()

Returns loss coefficient for any exit to a pipe as shown in *[R291]* and in other sources.

$$K = 1$$

Returns **K** : float

Loss coefficient [-]

Notes

It has been found on occasion that K = 2.0 for laminar flow, and ranges from about 1.04 to 1.10 for turbulent flow.

References

[R291]

Examples

```
>>> exit_normal()
1.0
```

fluids.fittings.**bend_rounded**(*Di=None*, *rc=None*, *angle=None*, *fd=None*, *bend_diameters=5*)

Returns loss coefficient for any rounded bend in a pipe as shown in *[R292]*.

$$K = f\alpha\frac{r}{d} + (0.10 + 2.4f)\sin(\alpha/2) + \frac{6.6f(\sqrt{\sin(\alpha/2)} + \sin(\alpha/2))}{(r/d)^{\frac{4\alpha}{\pi}}}$$

Parameters **Di** : float

Inside diameter of pipe, [m]

rc : float

Radius of curvatuce of the entrance, optional [m]

angle : float

Angle of bend, [degrees]

fd : float

Darcy friction factor [-]

bend_diameters : float

Number of diameters of pipe making up the bend radius [-]

Returns **K** : float

Loss coefficient [-]

Notes

When inputting bend diameters, note that manufacturers often specify this as a multiplier of nominal diameter, which is different than actual diameter. Those require that rc be specified.

First term represents surface friction loss; the second, secondary flows; and the third, flow separation. Encompasses the entire range of elbow and pipe bend configurations.

References

[R292]

Examples

```
>>> [bend_rounded(Di=4.020, rc=4.0*5, angle=i, fd=0.0163) for i in [15, 30, 45, 60, 75, 90]]
[0.07038212630028828, 0.10680196344492195, 0.13858204974134541, 0.16977191374717754, 0.201149415
>>> [bend_rounded(Di=34.500, rc=36*10, angle=i, fd=0.0106) for i in [15, 30, 45, 60, 75, 90]]
[0.061075866683922314, 0.10162621862720357, 0.14158887563243763, 0.18225270014527103, 0.22309967
```

fluids.fittings.**bend_miter**(*angle*)

Returns loss coefficient for any single-joint miter bend in a pipe as shown in *[R293]*.

$$K = 0.42 \sin(\alpha/2) + 2.56 \sin^3(\alpha/2)$$

Parameters **angle** : float

Angle of bend, [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

Applies for bends from 0 to 150 degrees. One joint only.

References

[R293]

Examples

```
>>> [bend_miter(i) for i in [150, 120, 90, 75, 60, 45, 30, 15]]
[2.7128147734758103, 2.0264994448555864, 1.2020815280171306, 0.8332188430731828, 0.5299999999999
```

fluids.fittings.**helix**(*Di=None*, *rs=None*, *pitch=None*, *N=None*, *fd=None*)

Returns loss coefficient for any size constant-pitch helix as shown in *[R294]*. Has applications in immersed coils in tanks.

$$K = N\left[f\frac{\sqrt{(2\pi r)^2 + p^2}}{d} + 0.20 + 4.8f\right]$$

Parameters **Di** : float

Inside diameter of pipe, [m]

rs : float

Radius of spiral, [m]

pitch : float

Distance between two subsequent coil centers, [m]

N : float

Number of coils in the helix [-]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

Formulation based on peak secondary flow as in two 180 degree bends per coil. Flow separation ignored. No f, Re, geometry limitations. Source not compared against others.

References

[R294]

Examples

```
>>> helix(Di=0.01, rs=0.1, pitch=.03, N=10, fd=.0185)
14.525134924495514
```

fluids.fittings.**spiral**(*Di=None*, *rmax=None*, *rmin=None*, *pitch=None*, *fd=None*)

Returns loss coefficient for any size constant-pitch spiral as shown in *[R295]*. Has applications in immersed coils in tanks.

$$K = \frac{r_{max} - r_{min}}{p}\left[f\pi\left(\frac{r_{max} + r_{min}}{d}\right) + 0.20 + 4.8f\right] + \frac{13.2f}{(r_{min}/d)^2}$$

Parameters **Di** : float

Inside diameter of pipe, [m]

rmax : float

Radius of spiral at extremity, [m]

rmax : float

Radius of spiral at end near center, [m]

pitch : float

Distance between two subsequent coil centers, [m]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

Source not compared against others.

References

[R295]

Examples

```
>>> spiral(Di=0.01, rmax=.1, rmin=.02, pitch=.01, fd=0.0185)
7.950918552775473
```

fluids.fittings.**Darby3K**(*NPS=None*, *Re=None*, *name=None*, *K1=None*, *Ki=None*, *Kd=None*)

Returns loss coefficient for any various fittings, depending on the name input. Alternatively, the Darby constants K1, Ki and Kd may be provided and used instead. Source of data is *[R296]*. Reviews of this model are favorable.

$$K_f = \frac{K_1}{Re} + K_i\left(1 + \frac{K_d}{D_{\text{NPS}}^{0.3}}\right)$$

Parameters **NPS** : float

Nominal diameter of the pipe, [in]

Re : float

Reynolds number, [-]

name : str

String from Darby dict representing a fitting

K1 : float

K1 parameter of Darby model, optional [-]

Ki : float

Ki parameter of Darby model, optional [-]

Kd : float

Kd parameter of Darby model, optional [in]

Returns **K** : float

Loss coefficient [-]

Notes

Also described in Albright's Handbook and Ludwig's Applied Process Design. Relatively uncommon to see it used.

The possibility of combining these methods with those above are attractive.

References

[R296], *[R297]*

Examples

```
>>> Darby3K(NPS=2., Re=10000., name='Valve, Angle valve, 45°, full line size, β = 1')
1.1572523963562353
>>> Darby3K(NPS=12., Re=10000., name='Valve, Angle valve, 45°, full line size, β = 1')
0.819510280626355
>>> Darby3K(NPS=12., Re=10000., K1=950,  Ki=0.25,  Kd=4)
0.819510280626355
```

fluids.fittings.**Hooper2K**(*Di=None*, *Re=None*, *name=None*, *K1=None*, *Kinfty=None*)

Returns loss coefficient for any various fittings, depending on the name input. Alternatively, the Hooper constants K1, Kinfty may be provided and used instead. Source of data is *[R298]*. Reviews of this model are favorable less favorable than the Darby method but superior to the constant-K method.

$$K = \frac{K_1}{Re} + K_\infty \left(1 + \frac{1}{ID_{in}}\right)$$

Parameters **Di** : float

Actual inside diameter of the pipe, [in]

Re : float

Reynolds number, [-]

name : str

String from Hooper dict representing a fitting

K1 : float

K1 parameter of Hooper model, optional [-]

Kinfty : float

Kinfty parameter of Hooper model, optional [-]

Returns **K** : float

Loss coefficient [-]

Notes

Also described in Ludwig's Applied Process Design. Relatively uncommon to see it used. No actual example found.

References

[R298], *[R299]*, *[R300]*

Examples

```
>>> Hooper2K(Di=2., Re=10000., name='Valve, Globe, Standard')
6.15
>>> Hooper2K(Di=2.067, Re=500., name='Valve, Check, Tilting-disc')
2.7418964683115625
```

fluids.fittings.**Kv_to_Cv**(*Kv*)

Convert valve flow coefficient from imperial to common metric units.

$$C_v = 1.1560992283540599 K_v$$

Parameters **Kv** : float

Valve flow coefficient, [1 m^3 cold water/hour at dP = 1 bar]

Returns **Cv** : float

Valve flow coefficient, [1 gpm water at 1.0 psi dP]

Notes

Kv = 0.865 Cv is in the IEC standard 60534-2-1. It has also been said that Cv = 1.17Kv; this is wrong by current standards.

References

[R301]

Examples

```
>>> Kv_to_Cv(2)
2.3121984567081197
```

fluids.fittings.**Cv_to_Kv**(*Cv*)

Convert valve flow coefficient from imperial to common metric units.

$$K_v = C_v/1.156$$

Parameters **Cv** : float

Valve flow coefficient, [1 gpm water at 1.0 psi dP]

Returns **Kv** : float

Valve flow coefficient, [1 m^3 cold water/hour at dP = 1 bar]

Notes

Kv = 0.865 Cv is in the IEC standard 60534-2-1. It has also been said that Cv = 1.17Kv; this is wrong by current standards.

References

[R302]

Examples

```
>>> Cv_to_Kv(2.312)
1.9998283393819036
```

fluids.fittings.**Kv_to_K**(*Kv*, *D*)

Convert valve flow coefficient from common metric units to regular loss coefficients.

$$K = 0.001604\frac{D^4}{K_v^2}$$

Parameters **Kv** : float

Valve flow coefficient, [1 m^3 cold water/hour at dP = 1 bar]

Returns **K** : float

Loss coefficient, [-]

References

[R303]

Examples

```
>>> Kv_to_K(2.312, .015)
15.1912580369009
```

fluids.friction_factor module

Chemical Engineering Design Library (ChEDL). Utilities for process modeling. Copyright (C) 2016, Caleb Bell <Caleb.Andrew.Bell@gmail.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

fluids.friction_factor.**friction_factor**(*Re=100000.0*, *eD=0.0001*, *Method=None*, *Darcy=True*, *AvailableMethods=False*)

Calculates friction factor. Uses a specified method, or automatically picks one from the dictionary of available methods. 28 methods available, described in the table below. The default is more than sufficient for all applications. Can also be accesed under the name *fd*.

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness of the wall, []

Returns **f** : float

Friction factor, [-]

methods : list, only returned if AvailableMethods == True

List of methods which claim to be valid for the range of *Re* and *eD* given

Other Parameters **Method** : string, optional

A string of the function name to use

Darcy : bool, optional

If False, will return fanning friction factor, 1/4 of the Darcy value

AvailableMethods : bool, optional

If True, function will consider which methods claim to be valid for the range of *Re* and *eD* given

Notes

Nice name	Re min	Re max	Re Default	ϵ/D Min	ϵ/D Max	ϵ/D Default
Rao Kumar 2007	None	None	None	None	None	None
Eck 1973	None	None	None	None	None	None
Jain 1976	5000	1.0E+7	None	4.0E-5	0.05	None
Avci Karagoz 2009	None	None	None	None	None	None
Swamee Jain 1976	5000	1.0E+8	None	1.0E-6	0.05	None
Churchill 1977	None	None	None	None	None	None
Brkic 2011 1	None	None	None	None	None	None
Chen 1979	4000	4.0E+8	None	1.0E-7	0.05	None
Round 1980	4000	4.0E+8	None	0	0.05	None
Papaevangelo 2010	10000	1.0E+7	None	1.0E-5	0.001	None
Fang 2011	3000	1.0E+8	None	0	0.05	None
Shacham 1980	4000	4.0E+8	None	None	None	None
Barr 1981	None	None	None	None	None	None
Churchill 1973	None	None	None	None	None	None
Moody	4000	1.0E+8	None	0	1	None
Zigrang Sylvester 1	4000	1.0E+8	None	4.0E-5	0.05	None
Zigrang Sylvester 2	4000	1.0E+8	None	4.0E-5	0.05	None
Buzzelli 2008	None	None	None	None	None	None
Haaland	4000	1.0E+8	None	1.0E-6	0.05	None
Serghides 1	None	None	None	None	None	None
Serghides 2	None	None	None	None	None	None
Tsal 1989	4000	1.0E+8	None	0	0.05	None
Alshul 1952	None	None	None	None	None	None
Wood 1966	4000	5.0E+7	None	1.0E-5	0.04	None
Manadilli 1997	5245	1.0E+8	None	0	0.05	None
Brkic 2011 2	None	None	None	None	None	None
Romeo 2002	3000	1.5E+8	None	0	0.05	None
Sonnad Goudar 2006	4000	1.0E+8	None	1.0E-6	0.05	None

Examples

```
>>> friction_factor(Re=1E5, eD=1E-4)
0.018513948401365277
```

fluids.friction_factor.**Moody**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Moody (1947) as shown in *[R304]* and originally in *[R305]*.

$$f_f = 1.375 \times 10^{-3}\left[1 + \left(2 \times 10^4 \frac{\epsilon}{D} + \frac{10^6}{Re}\right)^{1/3}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is Re >= 4E3 and Re <= 1E8; eD >= 0 < 0.01.

References

[R304], *[R305]*

Examples

```
>>> Moody(1E5, 1E-4)
0.01809185666808665
```

fluids.friction_factor.**Alshul_1952**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Alshul (1952) as shown in *[R306]*.

$$f_d = 0.11\left(\frac{68}{Re} + \frac{\epsilon}{D}\right)^{0.25}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R306]

Examples

```
>>> Alshul_1952(1E5, 1E-4)
0.018382997825686878
```

fluids.friction_factor.**Wood_1966**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Wood (1966) *[R308]* as shown in *[R307]*.

$$f_d = 0.094(\frac{\epsilon}{D})^{0.225} + 0.53(\frac{\epsilon}{D}) + 88(\frac{\epsilon}{D})^{0.4} Re^{-A_1}$$
$$A_1 = 1.62(\frac{\epsilon}{D})^{0.134}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 5E7; 1E-5 <= eD <= 4E-2.

References

[R307], *[R308]*

Examples

```
>>> Wood_1966(1E5, 1E-4)
0.021587570560090762
```

fluids.friction_factor.**Churchill_1973**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Churchill (1973) *[R310]* as shown in *[R309]*

$$\frac{1}{\sqrt{f_d}} = -2\log\left[\frac{\epsilon}{3.7D} + (\frac{7}{Re})^{0.9}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R309], *[R310]*

Examples

```
>>> Churchill_1973(1E5, 1E-4)
0.01846708694482294
```

fluids.friction_factor.**Eck_1973**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Eck (1973) *[R312]* as shown in *[R311]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left[\frac{\epsilon}{3.715D} + \frac{15}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R311], *[R312]*

Examples

```
>>> Eck_1973(1E5, 1E-4)
0.01775666973488564
```

fluids.friction_factor.**Jain_1976**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Jain (1976) *[R314]* as shown in *[R313]*.

$$\frac{1}{\sqrt{f_f}} = 2.28 - 4\log\left[\frac{\epsilon}{D} + \left(\frac{29.843}{Re}\right)^{0.9}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 5E3 <= Re <= 1E7; 4E-5 <= eD <= 5E-2.

References

[R313], *[R314]*

Examples

```
>>> Jain_1976(1E5, 1E-4)
0.018436560312693327
```

fluids.friction_factor.**Swamee_Jain_1976**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Swamee and Jain (1976) *[R316]* as shown in *[R315]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\left(\frac{6.97}{Re}\right)^{0.9} + (\frac{\epsilon}{3.7D})\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 5E3 <= Re <= 1E8; 1E-6 <= eD <= 5E-2.

References

[R315], *[R316]*

Examples

```
>>> Swamee_Jain_1976(1E5, 1E-4)
0.018452424431901808
```

fluids.friction_factor.**Churchill_1977**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Churchill and (1977) *[R318]* as shown in *[R317]*.

$$f_f = 2\left[(\frac{8}{Re})^{12} + (A_2 + A_3)^{-1.5}\right]^{1/12}$$

$$A_2 = \left\{2.457\ln\left[(\frac{7}{Re})^{0.9} + 0.27\frac{\epsilon}{D}\right]\right\}^{16}$$

$$A_3 = \left(\frac{37530}{Re}\right)^{16}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R317], *[R318]*

Examples

```
>>> Churchill_1977(1E5, 1E-4)
0.018462624566280075
```

fluids.friction_factor.**Chen_1979**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Chen (1979) *[R320]* as shown in *[R319]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\frac{\epsilon}{3.7065D} - \frac{5.0452}{Re}\log A_4\right]$$

$$A_4 = \frac{(\epsilon/D)^{1.1098}}{2.8257} + \left(\frac{7.149}{Re}\right)^{0.8981}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 4E8; 1E-7 <= eD <= 5E-2.

References

[R319], *[R320]*

Examples

```
>>> Chen_1979(1E5, 1E-4)
0.018552817507472126
```

fluids.friction_factor.**Round_1980**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Round (1980) *[R322]* as shown in *[R321]*.

$$\frac{1}{\sqrt{f_f}} = -3.6 \log\left[\frac{Re}{0.135 Re \frac{\epsilon}{D} + 6.5}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 4E8; 0 <= eD <= 5E-2.

References

[R321], *[R322]*

Examples

```
>>> Round_1980(1E5, 1E-4)
0.01831475391244354
```

fluids.friction_factor.**Shacham_1980**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Shacham (1980) *[R324]* as shown in *[R323]*.

$$\frac{1}{\sqrt{f_f}} = -4 \log\left[\frac{\epsilon}{3.7D} - \frac{5.02}{Re} \log\left(\frac{\epsilon}{3.7D} + \frac{14.5}{Re}\right)\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 4E8

References

[R323], *[R324]*

Examples

```
>>> Shacham_1980(1E5, 1E-4)
0.01860641215097828
```

fluids.friction_factor.**Barr_1981**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Barr (1981) *[R326]* as shown in *[R325]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left\{\frac{\epsilon}{3.7D} + \frac{4.518\log(\frac{Re}{7})}{Re\left[1+\frac{Re^{0.52}}{29}\left(\frac{\epsilon}{D}\right)^{0.7}\right]}\right\}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R325], *[R326]*

Examples

```
>>> Barr_1981(1E5, 1E-4)
0.01849836032779929
```

fluids.friction_factor.**Zigrang_Sylvester_1**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Zigrang and Sylvester (1982) *[R328]* as shown in *[R327]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\frac{\epsilon}{3.7D} - \frac{5.02}{Re}\log A_5\right]$$
$$A_5 = \frac{\epsilon}{3.7D} + \frac{13}{Re}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 4E-5 <= eD <= 5E-2.

References

[R327], *[R328]*

Examples

```
>>> Zigrang_Sylvester_1(1E5, 1E-4)
0.018646892425980794
```

`fluids.friction_factor.`**`Zigrang_Sylvester_2`**(*Re*, *eD*)

Calculates Darcy friction factor using the second method in Zigrang and Sylvester (1982) *[R330]* as shown in *[R329]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\frac{\epsilon}{3.7D} - \frac{5.02}{Re}\log A_6\right]$$
$$A_6 = \frac{\epsilon}{3.7D} - \frac{5.02}{Re}\log A_5$$
$$A_5 = \frac{\epsilon}{3.7D} + \frac{13}{Re}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 4E-5 <= eD <= 5E-2

References

[R329], *[R330]*

Examples

```
>>> Zigrang_Sylvester_2(1E5, 1E-4)
0.01850021312358548
```

fluids.friction_factor.**Haaland**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Haaland (1983) *[R332]* as shown in *[R331]*.

$$f_f = \left(-1.8\log_{10}\left[\left(\frac{\epsilon/D}{3.7}\right)^{1.11} + \frac{6.9}{Re}\right]\right)^{-2}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 1E-6 <= eD <= 5E-2

References

[R331], *[R332]*

Examples

```
>>> Haaland(1E5, 1E-4)
0.018265053014793857
```

fluids.friction_factor.**Serghides_1**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Serghides (1984) *[R334]* as shown in *[R333]*.

$$f = \left[A - \frac{(B-A)^2}{C - 2B + A}\right]^{-2}$$
$$A = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{12}{Re}\right]$$
$$B = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51A}{Re}\right]$$
$$C = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51B}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R333], *[R334]*

Examples

```
>>> Serghides_1(1E5, 1E-4)
0.01851358983180063
```

fluids.friction_factor.**Serghides_2**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Serghides (1984) *[R336]* as shown in *[R335]*.

$$f_d = \left[4.781 - \frac{(A-4.781)^2}{B - 2A + 4.781}\right]^{-2}$$
$$A = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{12}{Re}\right]$$
$$B = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51A}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R335], *[R336]*

Examples

```
>>> Serghides_2(1E5, 1E-4)
0.018486377560664482
```

fluids.friction_factor.**Tsal_1989**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Tsal (1989) *[R338]* as shown in *[R337]*.

$$A = 0.11(\frac{68}{Re} + \frac{\epsilon}{D})^{0.25}$$

if A >= 0.018 then fd = A if A < 0.018 then fd = 0.0028 + 0.85 A

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 0 <= eD <= 5E-2

References

[R337], *[R338]*

Examples

```
>>> Tsal_1989(1E5, 1E-4)
0.018382997825686878
```

fluids.friction_factor.**Manadilli_1997**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Manadilli (1997) *[R340]* as shown in *[R339]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left[\frac{\epsilon}{3.7D} + \frac{95}{Re^{0.983}} - \frac{96.82}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 5.245E3 <= Re <= 1E8; 0 <= eD <= 5E-2

References

[R339], *[R340]*

Examples

```
>>> Manadilli_1997(1E5, 1E-4)
0.01856964649724108
```

fluids.friction_factor.**Romeo_2002**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Romeo (2002) *[R342]* as shown in *[R341]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left\{\frac{\epsilon}{3.7065D} \times \frac{5.0272}{Re} \times \log\left[\frac{\epsilon}{3.827D} - \frac{4.567}{Re} \times \log\left(\frac{\epsilon}{7.7918D}^{0.9924} + \left(\frac{5.3326}{208.815 + Re}\right)^{0.9345}\right)\right]\right\}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 3E3 <= Re <= 1.5E8; 0 <= eD <= 5E-2

References

[R341], *[R342]*

Examples

```
>>> Romeo_2002(1E5, 1E-4)
0.018530291219676177
```

fluids.friction_factor.**Sonnad_Goudar_2006**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Sonnad and Goudar (2006) *[R344]* as shown in *[R343]*.

$$\frac{1}{\sqrt{f_d}} = 0.8686 \ln\left(\frac{0.4587Re}{S^{S/(S+1)}}\right)$$

$$S = 0.1240 \times \frac{\epsilon}{D} \times Re + \ln(0.4587Re)$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 1E-6 <= eD <= 5E-2

References

[R343], *[R344]*

Examples

```
>>> Sonnad_Goudar_2006(1E5, 1E-4)
0.0185971269898162
```

fluids.friction_factor.**Rao_Kumar_2007**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Rao and Kumar (2007) *[R346]* as shown in *[R345]*.

$$\frac{1}{\sqrt{f_d}} = 2\log\left(\frac{(2\frac{\epsilon}{D})^{-1}}{\left(\frac{0.444+0.135Re}{Re}\right)\beta}\right)$$

$$\beta = 1 - 0.55\exp(-0.33\ln\left[\frac{Re}{6.5}\right]^2)$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation. This equation is fit to original experimental friction factor data. Accordingly, this equation should not be used unless appropriate consideration is given.

References

[R345], *[R346]*

Examples

```
>>> Rao_Kumar_2007(1E5, 1E-4)
0.01197759334600925
```

fluids.friction_factor.**Buzzelli_2008**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Buzzelli (2008) *[R348]* as shown in *[R347]*.

$$\frac{1}{\sqrt{f_d}} = B_1 - \left[\frac{B_1 + 2\log(\frac{B_2}{Re})}{1 + \frac{2.18}{B_2}}\right]$$

$$B_1 = \frac{0.774\ln(Re) - 1.41}{1 + 1.32\sqrt{\frac{\epsilon}{D}}}$$

$$B_2 = \frac{\epsilon}{3.7D}Re + 2.51 \times B_1$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R347], *[R348]*

Examples

```
>>> Buzzelli_2008(1E5, 1E-4)
0.018513948401365277
```

fluids.friction_factor.**Avci_Karagoz_2009**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Avci and Karagoz (2009) *[R350]* as shown in *[R349]*.

$$f_D = \frac{6.4}{\left\{\ln(Re) - \ln\left[1 + 0.01Re\frac{\epsilon}{D}\left(1 + 10(\frac{\epsilon}{D})^{0.5}\right)\right]\right\}^{2.4}}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R349], *[R350]*

Examples

```
>>> Avci_Karagoz_2009(1E5, 1E-4)
0.01857058061066499
```

fluids.friction_factor.**Papaevangelo_2010**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Papaevangelo (2010) *[R352]* as shown in *[R351]*.

$$f_D = \frac{0.2479 - 0.0000947(7 - \log Re)^4}{\left[\log\left(\frac{\epsilon}{3.615D} + \frac{7.366}{Re^{0.9142}}\right)\right]^2}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 1E4 <= Re <= 1E7; 1E-5 <= eD <= 1E-3

References

[R351], *[R352]*

Examples

```
>>> Papaevangelo_2010(1E5, 1E-4)
0.015685600818488177
```

fluids.friction_factor.**Brkic_2011_1**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Brkic (2011) *[R354]* as shown in *[R353]*.

$$f_d = [-2\log(10^{-0.4343\beta} + \frac{\epsilon}{3.71D})]^{-2}$$

$$\beta = \ln\frac{Re}{1.816\ln\left(\frac{1.1Re}{\ln(1+1.1Re)}\right)}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R353], *[R354]*

Examples

```
>>> Brkic_2011_1(1E5, 1E-4)
0.01812455874141297
```

fluids.friction_factor.**Brkic_2011_2**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Brkic (2011) *[R356]* as shown in *[R355]*.

$$f_d = [-2\log(\frac{2.18\beta}{Re} + \frac{\epsilon}{3.71D})]^{-2}$$

$$\beta = \ln\frac{Re}{1.816\ln\left(\frac{1.1Re}{\ln(1+1.1Re)}\right)}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R355], *[R356]*

Examples

```
>>> Brkic_2011_2(1E5, 1E-4)
0.018619745410688716
```

fluids.friction_factor.**Fang_2011**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Fang (2011) *[R358]* as shown in *[R357]*.

$$f_D = 1.613\left\{\ln\left[0.234\frac{\epsilon}{D}^{1.1007} - \frac{60.525}{Re^{1.1105}} + \frac{56.291}{Re^{1.0712}}\right]\right\}^{-2}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 3E3 <= Re <= 1E8; 0 <= eD <= 5E-2

References

[R357], *[R358]*

Examples

```
>>> Fang_2011(1E5, 1E-4)
0.018481390682985432
```

fluids.geometry module

Chemical Engineering Design Library (ChEDL). Utilities for process modeling. Copyright (C) 2016, Caleb Bell <Caleb.Andrew.Bell@gmail.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

class `fluids.geometry.`**`TANK`**(*D=None*, *L=None*, *horizontal=True*, *sideA=None*, *sideB=None*, *sideA_a=0*, *sideB_a=0*, *sideA_f=1.0*, *sideA_k=0.06*, *sideB_f=1.0*, *sideB_k=0.06*, *sideA_a_ratio=0.25*, *sideB_a_ratio=0.25*, *L_over_D=None*, *V=None*)

Bases: `object`

Class representing tank volumes and levels. All parameters are also attributes.

Examples

Total volume of a tank:

```
>>> TANK(D=1.2, L=4, horizontal=False).V_total
4.523893421169302
```

Volume of a tank at a given height:

```
>>> TANK(D=1.2, L=4, horizontal=False).V_from_h(.5)
0.5654866776461628
```

Height of liquid for a given volume:

```
>>> TANK(D=1.2, L=4, horizontal=False).h_from_V(.5)
0.44209706414415373
```

Solving for tank volumes, first horizontal, then vertical:

```
>>> TANK(D=10., horizontal=True, sideA='conical', sideB='conical', V=500).L
4.699531057009146
>>> TANK(L=4.69953105701, horizontal=True, sideA='conical', sideB='conical', V=500).D
9.999999999999407
>>> TANK(L_over_D=0.469953105701, horizontal=True, sideA='conical', sideB='conical', V=500).L
4.69953105700979
```

```
>>> TANK(D=10., horizontal=False, sideA='conical', sideB='conical', V=500).L
4.699531057009147
>>> TANK(L=4.69953105701, horizontal=False, sideA='conical', sideB='conical', V=500).D
9.999999999999407
>>> TANK(L_over_D=0.469953105701, horizontal=False, sideA='conical', sideB='conical', V=500).L
4.69953105700979
```

Attributes

table	(bool) Whether or not a table of heights-volumes has been generated
h_max	(float) Height of the tank, [m]
V_total	(float) Total volume of the tank as calculated [m^3]
heights	(ndarray) Array of heights between 0 and h_max, [m]
volumes	(ndarray) Array of volumes calculated from the heights [m^3]

Methods

V_from_h(*h*)
Method to calculate the volume of liquid in a fully defined tank given a specified height *h*. *h* must be under the maximum height.

Parameters **h** : float

Height specified, [m]

Returns **V** : float

Volume of liquid in the tank up to the specified height, [m^3]

h_from_V(*V*)
Method to calculate the height of liquid in a fully defined tank given a specified volume of liquid in it *V*. *V* must be under the maximum volume. If interpolation table is not yet defined, creates it by calling the method set_table.

Parameters **V** : float

Volume of liquid in the tank up to the desired height, [m^3]

Returns **h** : float

Height of liquid at which the volume is as desired, [m]

set_misc()
Set more parameters, after the tank is better defined than in the __init__ function.

Notes

Two of D, L, and L_over_D must be known when this function runs. The other one is set from the other two first thing in this function. a_ratio parameters are used to calculate a values for the heads here, if applicable. Radius is calculated here. Maximum tank height is calculated here. V_total is calculated here.

set_table(*n=100*, *dx=None*)
Method to set an interpolation table of liquids levels versus volumes in the tank, for a fully defined tank. Normally run by the h_from_V method, this may be run prior to its use with a custom specification. Either the number of points on the table, or the vertical distance between steps may be specified.

Parameters **n** : float, optional

Number of points in the interpolation table, [-]

dx : float, optional

Vertical distance between steps in the interpolation table, [m]

solve_tank_for_V()
Method which is called to solve for tank geometry when a certain volume is specified. Will be called by the __init__ method if V is set.

Notes

Raises an error if L and either of sideA_a or sideB_a are specified; these can only be set once D is known. Raises an error if more than one of D, L, or L_over_D are specified. Raises an error if the head ratios are not provided.

Calculates initial guesses assuming no heads are present, and then uses fsolve to determine the correct dimentions for the tank.

Tested, but bugs and limitations are expected here.

table = False

fluids.mixing module

fluids.mixing.**adjust_homogeneity**(*fraction*)
Base: 95% homogeneity

fluids.mixing.**agitator_time_homogeneous**(*D=None*, *N=None*, *P=None*, *T=None*, *H=None*, *mu=None*, *rho=None*, *homogeneity=0.95*)
Calculates time for a fluid mizing in a tank with an impeller to reach a specified level of homogeneity, according to *[R359]*.

$$N_p = \frac{Pg}{\rho N^3 D^5}$$

$$Re_{imp} = \frac{\rho D^2 N}{\mu}$$

$$\text{constant} = N_p^{1/3} Re_{imp}$$

$$Fo = 5.2/\text{constant} \text{for turbulent regime}$$

$$Fo = (183/\text{constant})^2 \text{for transition regime}$$

Parameters **D** : float

Impeller diameter (optional) [m]

N : float:

Speed of impeller, [r/s]

P : float

Actual power required to mix, ignoring mechanical inefficiencies [W]

T : float

Tank diameter, [m]

H : float

Tank height, [m]

mu : float

Mixture viscosity, [Pa*s]

rho : float

Mixture density, [kg/m^3]

homogeneity : float

Fraction completion of mixing, optional, []

Returns **t** : float

Time for specified degree of homogeneity [s]

Notes

If impeller diameter is not specified, assumed to be 0.5 tank diameters.

The first example is solved forward rather than backwards here. A rather different result is obtained, but is accurate.

No check to see if the mixture if laminar is currently implemented. This would underpredict the required time.

References

[R359]

Examples

```
>>> agitator_time_homogeneous(D=36*.0254, N=56/60., P=957., T=1.83, H=1.83, mu=0.018, rho=1020,
15.143198226374668
```

```
>>> agitator_time_homogeneous(D=1, N=125/60., P=298., T=3, H=2.5, mu=.5, rho=980, homogeneity=.9
67.7575069865228
```

fluids.mixing.**Kp_helical_ribbon_Rieger**(*D=None*, *h=None*, *nb=None*, *pitch=None*, *width=None*, *T=None*)

Calculates product of power number and reynolds number for a specified geometry for a heilical ribbon mixer in the laminar regime. One of several correlations listed in *[R360]*, it used more data than other listed correlations and was recommended.

$$K_p = 82.8\frac{h}{D}\left(\frac{c}{D}\right)^{-0.38}\left(\frac{p}{D}\right)^{-0.35}\left(\frac{w}{D}\right)^{0.20} n_b^{0.78}$$

Parameters **D** : float

Impeller diameter (optional) [m]

h : float

Ribbon mixer height, [m]

nb : float:

Number of blades, [-]

pitch : float

Height of one turn around a helix [m]

width : float

Width of one blade [m]

T : float

Tank diameter, [m]

Returns **Kp** : float

Product of power number and reynolds number for laminar regime []

Notes

Example is from example 9-6 in *[R360]*. Confirmed.

References

[R360], *[R361]*

Examples

```
>>> Kp_helical_ribbon_Rieger(D=1.9, h=1.9, nb=2, pitch=1.9, width=.19, T=2)
357.39749163259256
```

fluids.mixing.**time_helical_ribbon_Grenville**(*Kp*, *N*)

Calculates product of time required for mixing in a helical ribbon coil in the laminar regime according to the Grenville *[R363]* method recommended in *[R362]*.

$$t = 896 \times 10^3 K_p^{-1.69}/N$$

Parameters **Kp** : float

Product of power number and reynolds number for laminar regime []

N : float:

Speed of impeller, [r/s]

Returns **t** : float

Time for homogeneity [s]

Notes

Degree of homogeneity is not specified. Example is from example 9-6 in *[R362]*. Confirmed.

References

[R362], *[R363]*

Examples

```
>>> time_helical_ribbon_Grenville(357.4, 4/60.)
650.980654028894
```

fluids.mixing.**size_tee**(*Q1=None*, *Q2=None*, *D=None*, *D2=None*, *n=1*, *pipe_diameters=5*)

Calculates CoV of an optimal or specified tee for mixing at a tee according to *[R364]*. Assumes turbulent flow. The smaller stream in injected into the main pipe, which continues straight. COV calculation is according to *[R365]*.

$$TODO$$

Parameters **Q1** : float

Volumetric flow rate of larger stream [m^3/s]

Q2 : float

Volumetric flow rate of smaller stream [m^3/s]

D : float

Diameter of pipe after tee [m]

D2 : float

Diameter of mixing inlet, optional (optimally calculated if not specified) [m]

n : float

Number of jets, 1 to 4 []

pipe_diameters : float

Number of diameters along tail pipe for CoV calculation, 0 to 5 []

Returns **CoV** : float

Standard deviation of dimentionless concentration [-]

Notes

Not specified if this works for liquid also, though probably not. Example is from example Example 9-6 in *[R364]*. Low precision used in example.

References

[R364], *[R365]*

Examples

```
>>> size_tee(Q1=11.7, Q2=2.74, D=0.762, D2=None, n=1, pipe_diameters=5)
0.2940930233038544
```

fluids.mixing.**COV_motionless_mixer**(*Ki=None*, *Q1=None*, *Q2=None*, *pipe_diameters=None*)

Calculates CoV of a motionless mixer with a regression parameter in *[R366]* and originally in *[R367]*.

$$\frac{CoV}{CoV_0} = K_i^{L/D}$$

Parameters **Ki** : float

Correlation parameter specific to a mixer's design, [-]

Q1 : float

Volumetric flow rate of larger stream [m^3/s]

Q2 : float

Volumetric flow rate of smaller stream [m^3/s]

pipe_diameters : float

Number of diameters along tail pipe for CoV calculation, 0 to 5 []

Returns **CoV** : float

Standard deviation of dimentionless concentration [-]

Notes

Example 7-8.3.2 in *[R366]*, solved backwards.

References

[R366], *[R367]*

Examples

```
>>> COV_motionless_mixer(Ki=.33, Q1=11.7, Q2=2.74, pipe_diameters=4.74/.762)
0.0020900028665727685
```

fluids.mixing.**K_motionless_mixer**(*K=None*, *L=None*, *D=None*, *fd=None*)

Calculates loss ciefficient of a motionless mixer with a regression parameter in *[R368]* and originally in *[R369]*.

$$K = K_{L/T} f \frac{L}{D}$$

Parameters **K** : float

Correlation parameter specific to a mixer's design, [-] Also specific to laminar or turbulent regime.

L : float

Length of the motionless mixer [m]

D : float

Diameter of pipe [m]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient of mixer [-]

Notes

Related to example 7-8.3.2 in *[R368]*.

References

[R368], *[R369]*

Examples

```
>>> K_motionless_mixer(K=150, L=.762*5, D=.762, fd=.01)
7.5
```

fluids.open_flow module

`fluids.open_flow.`**`Q_weir_V_Shen`**(*h1*, *angle=90*)

Calculates the flow rate across a V-notch (triangular) weir from the height of the liquid above the tip of the notch, and with the angle of the notch. Most of these type of weir are 90 degrees. Model from *[R370]* as reproduced in *[R371]*.

Flow rate is given by:

$$Q = C \tan\left(\frac{\theta}{2}\right) \sqrt{g}(h_1 + k)^{2.5}$$

Parameters **h1** : float

Height of the fluid above the notch [m]

angle : float, optional

Angle of the notch [degrees]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

angles = [20, 40, 60, 80, 100] Cs = [0.59, 0.58, 0.575, 0.575, 0.58] k = [0.0028, 0.0017, 0.0012, 0.001, 0.001]

The following limits apply to the use of this equation:

h1 >= 0.05 m h2 > 0.45 m h1/h2 <= 0.4 m b > 0.9 m

$$\frac{h_1}{b} \tan\left(\frac{\theta}{2}\right) < 2$$

Flows are lower than obtained by the curves at http://www.lmnoeng.com/Weirs/vweir.php.

References

[R370], *[R371]*

Examples

```
>>> Q_weir_V_Shen(0.6, angle=45)
0.21071725775478228
>>> Q_weir_V_Shen(1.2)
2.8587083148501078
```

fluids.open_flow.**Q_weir_rectangular_Smith**(*h1*, *b*, *h2=None*, *b1=None*, *V1=None*)

fluids.open_flow.**Q_weir_rectangular_Kindsvater_Carter**(*h1*, *h2*, *b*)

Calculates the flow rate across rectangular weir from the height of the liquid above the crest of the notch, the liquid depth beneath it, and the width of the notch. Model from *[R372]* as reproduced in *[R373]*.

Flow rate is given by:

$$Q = 0.554\left(1 - 0.0035\frac{h_1}{h_2}\right)(b + 0.0025)\sqrt{g}(h_1 + 0.0001)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the rectangular flow section of the weir [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

b/b1 0.2 h1/h2 < 2 b > 0.15 m h1 > 0.03 m h2 > 0.1 m

References

[R372], *[R373]*

Examples

```
>>> Q_weir_rectangular_Kindsvater_Carter(0.2, 0.5, 1)
0.15545928949179422
```

fluids.open_flow.**Q_weir_rectangular_SIA**(*h1*, *h2*, *b*, *b1*)

Calculates the flow rate across rectangular weir from the height of the liquid above the crest of the notch, the liquid depth beneath it, and the width of the notch. Model from *[R374]* as reproduced in *[R375]*.

Flow rate is given by:

$$Q = 0.544\left[1 + 0.064\left(\frac{b}{b_1}\right)^2 + \frac{0.00626 - 0.00519(b/b_1)^2}{h_1 + 0.0016}\right]\left[1 + 0.5\left(\frac{b}{b_1}\right)^4\left(\frac{h_1}{h_1 + h_2}\right)^2\right] b\sqrt{g}h^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the rectangular flow section of the weir [m]

b1 : float

Width of the full section of the channel [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

b/b1 0.2 h1/h2 < 2 b > 0.15 m h1 > 0.03 m h2 > 0.1 m

References

[R374], *[R375]*

Examples

```
>>> Q_weir_rectangular_SIA(0.2, 0.5, 1, 2)
1.0408858453811165
```

fluids.open_flow.**Q_weir_rectangular_full_Ackers**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R376]* as reproduced in *[R377]*, confirmed with *[R378]*.

Flow rate is given by:

$$Q = 0.564\left(1 + 0.150\frac{h_1}{h_2}\right) b\sqrt{g}(h_1 + 0.001)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

h1 > 0.02 m h2 > 0.15 m h1/h2 2.2

References

[R376], *[R377]*, *[R378]*

Examples

Example as in *[R378]*, matches. However, example is unlikely in practice.

```
>>> Q_weir_rectangular_full_Ackers(h1=0.9, h2=0.6, b=5)
9.251938159899948
```

```
>>> Q_weir_rectangular_full_Ackers(h1=0.3, h2=0.4, b=2)
0.6489618999846898
```

fluids.open_flow.**Q_weir_rectangular_full_SIA**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R379]* as reproduced in *[R380]*.

Flow rate is given by:

$$Q = \frac{2}{3}\sqrt{2}\left(0.615 + \frac{0.000615}{h_1 + 0.0016}\right) b\sqrt{g}h_1 + 0.5\left(\frac{h_1}{h_1 + h_2}\right)^2 b\sqrt{g}h_1^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

0.025 < h < 0.8 m b > 0.3 m h2 > 0.3 m h1/h2 < 1

References

[R379], *[R380]*

Examples

Example compares terribly with the Ackers expression - probable error in *[R380]*. DO NOT USE.

```
>>> Q_weir_rectangular_full_SIA(h1=0.3, h2=0.4, b=2)
1.1875825055400384
```

fluids.open_flow.**Q_weir_rectangular_full_Rehbock**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R382]* as reproduced in *[R383]*.

Flow rate is given by:

$$Q = \frac{2}{3}\sqrt{2}\left(0.602 + 0.0832\frac{h_1}{h_2}\right) b\sqrt{g}(h_1 + 0.00125)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

0.03 m < h1 < 0.75 m b > 0.3 m h2 > 0.3 m h1/h2 < 1

References

[R382], *[R383]*

Examples

```
>>> Q_weir_rectangular_full_Rehbock(h1=0.3, h2=0.4, b=2)
0.6486856330601333
```

fluids.open_flow.**Q_weir_rectangular_full_Kindsvater_Carter**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R384]* as reproduced in *[R385]*.

Flow rate is given by:

$$Q = \frac{2}{3}\sqrt{2}\left(0.602 + 0.0832\frac{h_1}{h_2}\right) b\sqrt{g}(h_1 + 0.00125)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

h1 > 0.03 m b > 0.15 m h2 > 0.1 m h1/h2 < 2

References

[R384], *[R385]*

Examples

```
>>> Q_weir_rectangular_full_Kindsvater_Carter(h1=0.3, h2=0.4, b=2)
0.641560300081563
```

fluids.open_flow.**V_Manning**(*Rh*, *S*, *n*)

Predicts the average velocity of a flow across an open channel of hydraulic radius Rh and slope S, given the Manning roughness coefficient n.

Flow rate is given by:

$$V = \frac{1}{n}R_h^{2/3}S^{0.5}$$

Parameters **Rh** : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

S : float

Slope of the channel, m/m [-]

n : float

Manning roughness coefficient [s/m^(1/3)]

Returns **V** : float

Average velocity of the channel [m/s]

Notes

This is equation is often given in imperial units multiplied by 1.49.

References

[R386], *[R387]*

Examples

Example is from *[R387]*, matches.

```
>>> V_Manning(0.2859, 0.005236, 0.03)*0.5721
0.5988618058239864
```

Custom example, checked.

```
>>> V_Manning(Rh=5, S=0.001, n=0.05)
1.8493111942973235
```

fluids.open_flow.**n_Manning_to_C_Chezy**(*n*, *Rh*)

Converts a Manning roughness coefficient to a Chezy coefficient, given the hydraulic radius of the channel.

$$C = \frac{1}{n} R_h^{1/6}$$

Parameters **n** : float

Manning roughness coefficient [s/m^(1/3)]

Rh : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

Returns **C** : float

Chezy coefficient [m^0.5/s]

References

[R388]

Examples

Custom example, checked.

```
>>> n_Manning_to_C_Chezy(0.05, Rh=5)
26.15320972023661
```

fluids.open_flow.**C_Chezy_to_n_Manning**(*C*, *Rh*)

Converts a Chezy coefficient to a Manning roughness coefficient, given the hydraulic radius of the channel.

$$n = \frac{1}{C} R_h^{1/6}$$

Parameters **C** : float

Chezy coefficient [m^0.5/s]

Rh : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

Returns **n** : float

Manning roughness coefficient [s/m^(1/3)]

References

[R389]

Examples

Custom example, checked.

```
>>> C_Chezy_to_n_Manning(26.15, Rh=5)
0.05000613713238358
```

fluids.open_flow.**V_Chezy**(*Rh*, *S*, *C*)

Predicts the average velocity of a flow across an open channel of hydraulic radius Rh and slope S, given the Chezy coefficient C.

Flow rate is given by:

$$V = C\sqrt{SR_h}$$

Parameters **Rh** : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

S : float

Slope of the channel, m/m [-]

C : float

Chezy coefficient [m^0.5/s]

Returns **V** : float

Average velocity of the channel [m/s]

References

[R390], *[R391]*, *[R392]*

Examples

Custom example, checked.

```
>>> V_Chezy(Rh=5, S=0.001, C=26.153)
1.8492963648371776
```

fluids.packed_bed module

`fluids.packed_bed.`**`Ergun`**(*Dp*, *voidage=0.4*, *sphericity=1*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed, using the famous Ergun equation.

Pressure drop is given by:

$$\Delta p = \frac{150\mu(1-\epsilon)^2 V_s L}{\epsilon^3(\Phi D_p)^2} + \frac{1.75(1-\epsilon)\rho V_s^2 L}{\epsilon^3(\Phi D_p)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

sphericity : float

Sphericity of particles in bed []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

The first term in this equation represents laminar loses, and the second, turbulent loses. Sphericity must be calculated. According to *[R393]*, developed using spheres, pulverized coke/coal, sand, cylinders and tablets for ranges of $1 < RE_{ERg} < 2300$. *[R393]* cites a source claiming it should not be used above 500.

References

[R393]

Examples

```
>>> # Custom example
>>> Ergun(Dp=0.0008, voidage=0.4, sphericity=1., H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-00
892.3013355913797
```

fluids.packed_bed.**Kuo_Nydegger**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R395]*, originally in *[R394]*.

Pressure drop is given by:

$$\frac{\Delta P}{L}\frac{D_p^2}{\mu v_s}\left(\frac{\phi}{1-\phi}\right)^2 = 276 + 5.05Re^{0.87}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

Does not share exact form with the Ergun equation. 767 < Re_{ergun} < 24330

References

[R394], *[R395]*

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Kuo_Nydegger(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
1658.3187666274648
```

Re = 4000 custom example:

```
>>> Kuo_Nydegger(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
241.56063171630015
```

fluids.packed_bed.**Jones_Krier**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R396]*.

Pressure drop is given by:

$$\frac{\Delta P}{L}\frac{D_p^2}{\mu v_s}\left(\frac{\phi}{1-\phi}\right)^2 = 150 + 1.89Re^{0.87}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

Does not share exact form with the Ergun equation. 733 < Re_{ergun} < 126670

References

[R396]

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Jones_Krier(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
911.494265366317
```

Re = 4000 custom example:

```
>>> Jones_Krier(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
184.69401245425462
```

fluids.packed_bed.**Carman**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R398]*, originally in *[R397]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = \frac{180}{Re_{Erg}} + \frac{2.87}{Re_{Erg}^{0.1}}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$0.1 < RE_{ERg} < 60000$

References

[R397], *[R398]*

Examples

Custom example:

```
>>> Carman(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
1077.0587868633704
```

Re = 4000 custom example:

```
>>> Carman(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
178.2490332160841
```

fluids.packed_bed.**Hicks**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R400]*, originally in *[R399]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = \frac{6.8}{Re_{Erg}^{0.2}}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$300 < RE_{ERg} < 60000$

References

[R399], *[R400]*

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Hicks(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
62.534899706155834
```

Re = 4000 custom example:

```
>>> Hicks(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
171.20579747453397
```

fluids.packed_bed.**Brauer**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R402]*, originally in *[R401]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = \frac{160}{Re_{Erg}} + \frac{3.1}{Re_{Erg}^{0.1}}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$0.01 < RE_{ERg} < 40000$

References

[R401], *[R402]*

Examples

Custom example:

```
>>> Brauer(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
962.7294566294247
```

Re = 4000 custom example:

```
>>> Brauer(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
191.77738833880164
```

fluids.packed_bed.**Montillet**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*, *Dc=None*)

Calculates pressure drop across a packed bed of spheres as in *[R404]*, originally in *[R403]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = a\left(\frac{D_c}{D_p}\right)^{0.20}\left(\frac{1000}{Re_p} + \frac{60}{Re_p^{0.5}} + 12\right)$$

$$Re_p = \frac{\rho v_s D_p}{\mu}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Dc : float, optional

Diameter of the column, [m]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$10 < REp < 2500$ if Dc/D > 50, set to 2.2. a = 0.061 for epsilon < 0.4, 0.050 for > 0.4.

References

[R403], *[R404]*

Examples

Custom example:

```
>>> Montillet(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
1148.1905244077548
```

Re = 4000 custom example:

```
>>> Montillet(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
212.67409611116554
```

fluids.packed_bed.**Idelchik**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R406]*, originally in *[R405]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p = \frac{0.765}{\epsilon^{4.2}}\left(\frac{30}{Re_l} + \frac{3}{Re_l^{0.7}} + 0.3\right)$$

$$Re_l = (0.45/\epsilon^{0.5}) Re_{Erg}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$0.001 < RE_{ERg} < 1000$ This equation is valid for void fractions between 0.3 and 0.8. Cited as by Bernshtein. This model is likely presented in *[R406]* with a typo, as it varries greatly from other models.

References

[R405], *[R406]*

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Idelchik(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
56.13431404382615
```

Re = 400 custom example:

```
>>> Idelchik(Dp=0.008, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
120.55068459098145
```

fluids.packed_bed.**voidage_Benyahia_Oneil**(*Dpe*, *Dt*, *sphericity*)

Calculates voidage of a bed of arbitraryily shaped uniform particles packed into a bed or tube of diameter *Dt*, with equivalent sphere diameter *Dp*. Shown in *[R407]*, and cited by various authors. Correlations exist also for spheres, solid cylinders, hollow cylinders, and 4-hole cylinders. Based on a series of physical measurements.

$$\epsilon = 0.1504 + \frac{0.2024}{\phi} + \frac{1.0814}{\left(\frac{d_t}{d_{pe}} + 0.1226\right)^2}$$

Parameters **Dpe** : float

Equivalent spherical particle diameter, [m]

Dt : float

Diameter of the tube, [m]

sphericity : float

Sphericity of particles in bed []

Returns **voidage** : float

Void fraction of bed packing []

Notes

Average error of 5.2%; valid 1.5 < dtube/dp < 50 and 0.42 < sphericity < 1

References

[R407]

Examples

```
>>> voidage_Benyahia_Oneil(1E-3, 1E-2, .8)
0.41395363849210065
```

fluids.packed_bed.**voidage_Benyahia_Oneil_spherical**(*Dp*, *Dt*)

Calculates voidage of a bed of spheres packed into a bed or tube of diameter *Dt*, with sphere diameters *Dp*. Shown in *[R408]*, and cited by various authors. Correlations exist also for solid cylinders, hollow cylinders, and 4-hole cylinders. Based on a series of physical measurements.

$$\epsilon = 0.390 + \frac{1.740}{\left(\frac{d_{cyl}}{d_p} + 1.140\right)^2}$$

Parameters **Dp** : float

Spherical particle diameter, [m]

Dt : float

Diameter of the tube, [m]

Returns **voidage** : float

Void fraction of bed packing []

Notes

Average error 1.5%, 1.5 < ratio < 50.

References

[R408]

Examples

```
>>> voidage_Benyahia_Oneil_spherical(.001, .05)
0.3906653157443224
```

fluids.packed_bed.**voidage_Benyahia_Oneil_cylindrical**(*Dpe*, *Dt*, *sphericity*)

Calculates voidage of a bed of cylindrical uniform particles packed into a bed or tube of diameter *Dt*, with equivalent sphere diameter *Dpe*. Shown in *[R409]*, and cited by various authors. Correlations exist also for spheres, solid cylinders, hollow cylinders, and 4-hole cylinders. Based on a series of physical measurements.

$$\epsilon = 0.373 + \frac{1.703}{\left(\frac{d_{cyl}}{d_p} + 0.611\right)^2}$$

Parameters **Dpe** : float

Equivalent spherical particle diameter, [m]

Dt : float

Diameter of the tube, [m]

sphericity : float

Sphericity of particles in bed []

Returns **voidage** : float

Void fraction of bed packing []

Notes

Average error 0.016%; 1.7 < ratio < 26.3.

References

[R409]

Examples

```
>>> voidage_Benyahia_Oneil_cylindrical(.01, .1, .6)
0.38812523109607894
```

fluids.piping module

`fluids.piping.`**`nearest_pipe`**(*Do=None*, *Di=None*, *NPS=None*, *schedule='40'*)

Searches for and finds the nearest standard pipe size to a given specification. Acceptable inputs are:

- Nominal pipe size
- Nominal pipe size and schedule
- Outer diameter *Do*
- Outer diameter *Do* and schedule
- Inner diameter *Di*
- Inner diameter *Di* and schedule

Acceptable schedules are: '5', '10', '20', '30', '40', '60', '80', '100', '120', '140', '160', 'STD', 'XS', 'XXS', '5S', '10S', '40S', '80S'.

Parameters **Do** : float

Pipe outer diameter, [m]

Di : float

Pipe inner diameter, [m]

NPS : float

Nominal pipe size, []

schedule : str

String representing schedule size

Returns **NPS** : float

Nominal pipe size, []

_di : float

Pipe inner diameter, [m]

_do : float

Pipe outer diameter, [m]

_t : float

Pipe wall thickness, [m]

Notes

Internal units within this function are mm. The imperial schedules are not quite identical to these value, but all rounding differences happen in the sub-0.1 mm level.

References

[R410], *[R411]*

Examples

```
>>> nearest_pipe(Di=0.021)
(1, 0.02664, 0.0334, 0.0033799999999999998)
>>> nearest_pipe(Do=.273, schedule='5S')
(10, 0.26630000000000004, 0.2731, 0.0034)
```

fluids.piping.**gauge_from_t**(*t*, *SI=True*, *schedule='BWG'*)

Looks up the gauge of a given wire thickness of given schedule. Values are all non-linear, and tabulated internally.

Parameters **t** : float

Thickness, [m]

SI : bool, optional

If False, value in inches is returned, rather than m.

schedule : str

Gauge schedule, one of 'BWG', 'AWG', 'SWG', 'MWG', 'BSWG', or 'SSWG'

Returns **gauge** : float-like

Wire Gauge, []

Notes

Birmingham Wire Gauge (BWG) ranges from 0.2 (0.5 inch) to 36 (0.004 inch).

American Wire Gauge (AWG) ranges from 0.167 (0.58 inch) to 51 (0.00099 inch). These are used for electrical wires.

Steel Wire Gauge (SWG) ranges from 0.143 (0.49 inch) to 51 (0.0044 inch). Also called Washburn & Moen wire gauge, American Steel gauge, Wire Co. gauge, and Roebling wire gauge.

Music Wire Gauge (MWG) ranges from 0.167 (0.004 inch) to 46 (0.18 inch). Also called Piano Wire Gauge.

British Standard Wire Gage (BSWG) ranges from 0.143 (0.5 inch) to 51 (0.001 inch). Also called Imperial Wire Gage (IWG).

Stub's Steel Wire Gage (SSWG) ranges from 1 (0.227 inch) to 80 (0.013 inch)

References

[R412]

Examples

```
>>> gauge_from_t(.5, False, 'BWG'), gauge_from_t(0.005588, True)
(0.2, 5)
>>> gauge_from_t(0.5165, False, 'AWG'), gauge_from_t(0.00462026, True, 'AWG')
(0.2, 5)
>>> gauge_from_t(.4305, False, 'SWG'), gauge_from_t(0.0052578, True, 'SWG')
(0.2, 5)
>>> gauge_from_t(.005, False, 'MWG'), gauge_from_t(0.0003556, True, 'MWG')
(0.2, 5)
>>> gauge_from_t(.432, False, 'BSWG'), gauge_from_t(0.0053848, True, 'BSWG')
(0.2, 5)
>>> gauge_from_t(0.227, False, 'SSWG'), gauge_from_t(0.0051816, True, 'SSWG')
(1, 5)
```

fluids.piping.**t_from_gauge**(*gauge*, *SI=True*, *schedule='BWG'*)

Looks up the thickness of a given wire gauge of given schedule. Values are all non-linear, and tabulated internally.

Parameters **gauge** : float-like

Wire Gauge, []

SI : bool, optional

If False, value in inches is returned, rather than m.

schedule : str

Gauge schedule, one of 'BWG', 'AWG', 'SWG', 'MWG', 'BSWG', or 'SSWG'

Returns **t** : float

Thickness, [m]

Notes

Birmingham Wire Gauge (BWG) ranges from 0.2 (0.5 inch) to 36 (0.004 inch).

American Wire Gauge (AWG) ranges from 0.167 (0.58 inch) to 51 (0.00099 inch). These are used for electrical wires.

Steel Wire Gauge (SWG) ranges from 0.143 (0.49 inch) to 51 (0.0044 inch). Also called Washburn & Moen wire gauge, American Steel gauge, Wire Co. gauge, and Roebling wire gauge.

Music Wire Gauge (MWG) ranges from 0.167 (0.004 inch) to 46 (0.18 inch). Also called Piano Wire Gauge.

British Standard Wire Gage (BSWG) ranges from 0.143 (0.5 inch) to 51 (0.001 inch). Also called Imperial Wire Gage (IWG).

Stub's Steel Wire Gage (SSWG) ranges from 1 (0.227 inch) to 80 (0.013 inch)

References

[R413]

Examples

```
>>> t_from_gauge(.2, False, 'BWG'), t_from_gauge(5, True)
(0.5, 0.005588)
>>> t_from_gauge(.2, False, 'AWG'), t_from_gauge(5, True, 'AWG')
(0.5165, 0.00462026)
>>> t_from_gauge(.2, False, 'SWG'), t_from_gauge(5, True, 'SWG')
(0.4305, 0.0052578)
>>> t_from_gauge(.2, False, 'MWG'), t_from_gauge(5, True, 'MWG')
(0.005, 0.0003556)
>>> t_from_gauge(.2, False, 'BSWG'), t_from_gauge(5, True, 'BSWG')
(0.432, 0.0053848)
>>> t_from_gauge(1, False, 'SSWG'), t_from_gauge(5, True, 'SSWG')
(0.227, 0.0051816)
```

fluids.pump module

Chemical Engineering Design Library (ChEDL). Utilities for process modeling. Copyright (C) 2016, Caleb Bell <Caleb.Andrew.Bell@gmail.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

`fluids.pump.`**`VFD_efficiency`**(*P*, *load=1*)

Returns the efficiency of a Variable Frequency Drive according to *[R414]*. These values are generic, and not standardized as minimum values. Older VFDs often have much worse performance.

Parameters **P** : float

Power, [W]

load : float, optional

Fraction of motor's rated electrical capacity being used

Returns **effciency** : float

VFD efficiency, [-]

Notes

The use of a VFD does change the characteristics of a pump curve's efficiency, but this has yet to be quantified. The effect is small. This value should be multiplied by the product of the pump and motor efficiency to determine the overall efficiency.

Efficiency table is in units of hp, so a conversion is performed internally. If load not specified, assumed 1 - where maximum efficiency occurs. Table extends down to 3 hp and up to 400 hp; values outside these limits are rounded to the nearest known value. Values between standardized sizes are interpolated linearly. Load values extend down to 0.016.

References

[R414]

Examples

```
>>> VFD_efficiency(10*hp)
0.96
>>> VFD_efficiency(100*hp, load=0.5)
0.96
```

fluids.pump.**CSA_motor_efficiency**(*P*, *closed=False*, *poles=2*, *high_efficiency=False*)

Returns the efficiency of a NEMA motor according to *[R415]*. These values are standards, but are only for full-load operation.

Parameters **P** : float

Power, [W]

closed : bool, optional

Whether or not the motor is enclosed

poles : int, optional

The number of poles of the motor

high_efficiency : bool, optional

Whether or not to look up the high-efficiency value

Returns **effciency** : float

Guaranteed full-load motor efficiency, [-]

Notes

Criteria for being required to meet the high-efficiency standard is:

- Designed for continuous operation
- Operates by three-phase induction
- Is a squirrel-cage or cage design
- Is NEMA type A, B, or C with T or U frame; or IEC design N or H
- Is designed for single-speed operation
- Has a nominal voltage of less than 600 V AC
- Has a nominal frequency of 60 Hz or 50/60 Hz
- Has 2, 4, or 6 pole construction
- Is either open or closed

Pretty much every motor is required to meet the low-standard efficiency table, however.

Several low-efficiency standard high power values were added to allow for easy programming; values are the last listed efficiency in the table.

References

[R415]

Examples

```
>>> CSA_motor_efficiency(100*hp)
0.93
>>> CSA_motor_efficiency(100*hp, closed=True, poles=6, high_efficiency=True)
0.95
```

fluids.pump.**motor_efficiency_underloaded**(*P*, *load=0.5*)

Returns the efficiency of a motor opperating under its design power according to *[R416]*.These values are generic; manufacturers usually list 4 points on their product information, but full-scale data is hard to find and not regulated.

Parameters **P** : float

Power, [W]

load : float, optional

Fraction of motor's rated electrical capacity being used

Returns **effciency** : float

Motor efficiency, [-]

Notes

If the efficiency returned by this function is unattractive, use a VFD. The curves used here are polynomial fits to *[R416]*'s graph, and curves were available for the following motor power ranges: 0-1 hp, 1.5-5 hp, 10 hp, 15-25 hp, 30-60 hp, 75-100 hp If above the upper limit of one range, the next value is returned.

References

[R416]

Examples

```
>>> motor_efficiency_underloaded(1*hp)
0.8705179600980149
>>> motor_efficiency_underloaded(10.1*hp,  .1)
0.6728425932357025
```

fluids.pump.**Corripio_pump_efficiency**(*Q*)

Estimates pump efficiency using the method in Corripio (1982) as shown in *[R417]* and originally in *[R418]*. Estimation only

$$\eta_P = -0.316 + 0.24015\ln(Q) - 0.01199\ln(Q)^2$$

Parameters **Q** : float

Volumetric flow rate, [m^3/s]

Returns **effciency** : float

Pump efficiency, [-]

Notes

For Centrifugal pumps only. Range is 50 to 5000 GPM, but input variable is in metric. Values above this range and below this range will go negative, although small deviations are acceptable. Example 16.5 in *[R417]*.

References

[R417], *[R418]*

Examples

```
>>> Corripio_pump_efficiency(461./15850.323)
0.7058888670951621
```

fluids.pump.**Corripio_motor_efficiency**(*P*)

Estimates motor efficiency using the method in Corripio (1982) as shown in *[R419]* and originally in *[R420]*. Estimation only.

$$\eta_M = 0.8 + 0.0319 \ln(P_B) - 0.00182 \ln(P_B)^2$$

Parameters **P** : float

Power, [W]

Returns **effciency** : float

Motor efficiency, [-]

Notes

Example 16.5 in *[R419]*.

References

[R419], *[R420]*

Examples

```
>>> Corripio_motor_efficiency(137*745.7)
0.9128920875679222
```

fluids.pump.**specific_speed**(*Q*, *H*, *n=3600.0*)

Returns the specific speed of a pump operating at a specified Q, H, and n.

$$n_S = \frac{n\sqrt{Q}}{H^{0.75}}$$

Parameters **Q** : float

Flow rate, [m^3/s]

H : float

Head generated by the pump, [m]

n : float, optional

Speed of pump [rpm]

Returns **nS** : float

Specific Speed, [rpm*m^0.75/s^0.5]

Notes

Defined at the BEP, with maximum fitting diameter impeller, at a given rotational speed.

References

[R421]

Examples

Example from *[R421]*.

```
>>> specific_speed(0.0402, 100, 3550)
22.50823182748925
```

fluids.pump.**specific_diameter**(*Q*, *H*, *D*)

Returns the specific diameter of a pump operating at a specified Q, H, and D.

$$D_s = \frac{DH^{1/4}}{\sqrt{Q}}$$

Parameters **Q** : float

Flow rate, [m^3/s]

H : float

Head generated by the pump, [m]

D : float

Pump impeller diameter [m]

Returns **Ds** : float

Specific diameter, [m^0.25/s^0.5]

Notes

Used in certain pump sizing calculations.

References

[R422]

Examples

```
>>> specific_diameter(Q=0.1, H=10., D=0.1)
0.5623413251903491
```

fluids.pump.**speed_synchronous**(*f*, *poles=2*, *phase=3*)
Returns the synchronous speed of a synchronous motor according to *[R423]*.

$$N_s = \frac{120 f \cdot \text{phase}}{\text{poles}}$$

Parameters **f** : float

Line frequency, [Hz]

poles : int, optional

The number of poles of the motor

phase : int, optional

Line AC phase

Returns **Ns** : float

Speed of synchronous motor, [rpm]

Notes

Synchronous motors have no slip. Large synchronous motors are not self-starting.

References

[R423]

Examples

```
>>> speed_synchronous(50, poles=12)
1500.0
>>> speed_synchronous(60, phase=1)
3600.0
```

fluids.pump.**nema_sizes = [186.42496789556753, 248.5666238607567, 372.84993579113507, 559.2749036867026, 745.6998**
list: all NEMA motor sizes, in Watts.

fluids.pump.**nema_sizes_hp = [0.25, 0.3333333333333333, 0.5, 0.75, 1, 1.5, 2, 3, 4, 5, 5.5, 7.5, 10, 15, 20, 25, 30, 40, 50, 60,**
list: all NEMA motor sizes, in hp.

fluids.pump.**motor_round_size**(*P*)
Rounds up the power for a motor to the nearest NEMA standard power. The returned power is always larger or equal to the input power.

Parameters **P** : float

Power, [W]

Returns **P_actual** : float

Actual power, equal to or larger than input [W]

Notes

An exception is raised if the power required is larger than any of the NEMA sizes. Larger motors are available, but are unstandardized.

References

[R424]

Examples

```
>>> [motor_round_size(i) for i in [.1*hp, .25*hp, 1E5, 3E5]]
[186.42496789556753, 186.42496789556753, 111854.98073734052, 335564.94221202156]
```

fluids.safety_valve module

Chemical Engineering Design Library (ChEDL). Utilities for process modeling. Copyright (C) 2016, Caleb Bell <Caleb.Andrew.Bell@gmail.com>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

fluids.safety_valve.**API520_round_size**(*A*)

Rounds up the area from an API 520 calculation to an API526 standard valve area. The returned area is always larger or equal to the input area.

Parameters **A** : float

Minimum discharge area [m^2]

Returns **area** : float

Actual discharge area [m^2]

Notes

To obtain the letter designation of an input area, lookup the area with the following:

API526_letters[API526_A.index(area)]

An exception is raised if the required relief area is larger than any of the API 526 sizes.

References

[R425]

Examples

From *[R425]*, checked with many points on Table 8.

```
>>> API520_round_size(1E-4)
0.00012645136
>>> API526_letters[API526_A.index(API520_round_size(1E-4))]
'E'
```

1.1.2 Module contents

`fluids.`**`T_critical_flow`**(*T*, *k*)

Calculates critical flow temperature *Tcf* for a fluid with the given isentropic coefficient. *Tcf* is in a flow (with Ma=1) whose stagnation conditions are known. Normally used with converging/diverging nozzles.

$$\frac{T^*}{T_0} = \frac{2}{k+1}$$

Parameters **T** : float

Stagnation temperature of a fluid with Ma=1 [K]

k : float

Isentropic coefficient []

Returns **Tcf** : float

Critical flow temperature at Ma=1 [K]

Notes

Assumes isentropic flow.

References

[R1]

Examples

Example 12.4 in *[R1]*:

```
>>> T_critical_flow(473, 1.289)
413.2809086937528
```

fluids.**P_critical_flow**(*P*, *k*)

Calculates critical flow pressure *Pcf* for a fluid with the given isentropic coefficient. *Pcf* is in a flow (with Ma=1) whose stagnation conditions are known. Normally used with converging/diverging nozzles.

$$\frac{P^*}{P_0} = \left(\frac{2}{k+1}\right)^{k/(k-1)}$$

Parameters **P** : float

Stagnation pressure of a fluid with Ma=1 [Pa]

k : float

Isentropic coefficient []

Returns **Pcf** : float

Critical flow pressure at Ma=1 [Pa]

Notes

Assumes isentropic flow.

References

[R2]

Examples

Example 12.4 in *[R2]*:

```
>>> P_critical_flow(1400000, 1.289)
766812.9022792266
```

fluids.**is_critical_flow**(*P1*, *P2*, *k*)

Determines if a flow of a fluid driven by pressure gradient P1 - P2 is critical, for a fluid with the given isentropic coefficient. This function calculates critical flow pressure, and checks if this is larger than P2. If so, the flow is critical and choked.

Parameters **P1: float**

Higher, source pressure [Pa]

P2: float

Lower, downstream pressure [Pa]

k : float

Isentropic coefficient []

Returns **flowtype** : bool

True if the flow is choked; otherwise False

Notes

Assumes isentropic flow. Uses P_critical_flow function.

References

[R3]

Examples

Examples 1-2 from API 520.

```
>>> is_critical_flow(670E3, 532E3, 1.11)
False
>>> is_critical_flow(670E3, 101E3, 1.11)
True
```

fluids.**stagnation_energy**(*V*)

Calculates the increase in enthalpy *dH* which is provided by a fluid's velocity *V*.

$$\Delta H = \frac{V^2}{2}$$

Parameters **V** : float

Velocity [m/s]

Returns **dH** : float

Incease in enthalpy [J/kg]

Notes

The units work out. This term is pretty small, but not trivial.

References

[R4]

Examples

```
>>> stagnation_energy(125)
7812.5
```

fluids.**P_stagnation**(*P*, *T*, *Tst*, *k*)

Calculates stagnation flow pressure *Pst* for a fluid with the given isentropic coefficient and specified stagnation temperature and normal temperature. Normally used with converging/diverging nozzles.

$$\frac{P_0}{P} = \left(\frac{T_0}{T}\right)^{\frac{k}{k-1}}$$

Parameters **P** : float

Normal pressure of a fluid [Pa]

T : float

Normal temperature of a fluid [K]

Tst : float

Stagnation temperature of a fluid moving at a certain velocity [K]

k : float

Isentropic coefficient []

Returns **Pst** : float

Stagnation pressure of a fluid moving at a certain velocity [Pa]

Notes

Assumes isentropic flow.

References

[R5]

Examples

Example 12-1 in *[R5]*.

```
>>> P_stagnation(54050., 255.7, 286.8, 1.4)
80772.80495900588
```

`fluids.`**`T_stagnation`**(*T*, *P*, *Pst*, *k*)

Calculates stagnation flow temperature *Tst* for a fluid with the given isentropic coefficient and specified stagnation pressure and normal pressure. Normally used with converging/diverging nozzles.

$$T = T_0 \left(\frac{P}{P_0}\right)^{\frac{k-1}{k}}$$

Parameters **T** : float

Normal temperature of a fluid [K]

P : float

Normal pressure of a fluid [Pa]

Pst : float

Stagnation pressure of a fluid moving at a certain velocity [Pa]

k : float

Isentropic coefficient []

Returns **Tst** : float

Stagnation temperature of a fluid moving at a certain velocity [K]

Notes

Assumes isentropic flow.

References

[R6]

Examples

Example 12-1 in *[R6]*.

```
>>> T_stagnation(286.8, 54050, 54050*8, 1.4)
519.5230938217768
```

fluids.**T_stagnation_ideal**(*T*, *V*, *Cp*)

Calculates the ideal stagnation temperature *Tst* calculated assuming the fluid has a constant heat capacity *Cp* and with a specified velocity *V* and tempeature *T*.

$$T^* = T + \frac{V^2}{2C_p}$$

Parameters **T** : float

Tempearture [K]

V : float

Velocity [m/s]

Cp : float

Ideal heat capacity [J/kg/K]

Returns **Tst** : float

Stagnation temperature [J/kg]

References

[R7]

Examples

Example 12-1 in *[R7]*.

```
>>> T_stagnation_ideal(255.7, 250, 1005.)
286.79452736318405
```

fluids.**cavitation_index**(*P1*, *P2*, *Psat*)

Calculates the cavitation index of a valve with upstream and downstream absolute pressures *P1* and *P2* for a fluid with a vapor pressure *Psat*.

$$\sigma = \frac{P_1 - P_{sat}}{P_1 - P_2}$$

Parameters **P1** : float

Absolute pressure upstream of the valve [Pa]

P2 : float

Absolute pressure downstream of the valve [Pa]

Psat : float

Saturation pressure of the liquid at inlet temperature [Pa]

Returns **sigma** : float

Cavitation index of the valve [-]

Notes

Larger values are safer. Models for adjusting cavitation indexes provided by the manufacturer to the user's conditions are available, making use of scaling the pressure differences and size differences.

Values can be calculated for incipient cavitation, constant cavitation, maximum vibration cavitation, incipient damage, and choking cavitation.

Has also been defined as:

$$\sigma = \frac{P_2 - P_{sat}}{P_1 - P_2}$$

Another definition and notation series is:

$$K = xF = \frac{1}{\sigma} = \frac{P_1 - P_2}{P_1 - P_{sat}}$$

References

[R8]

Examples

```
>>> cavitation_index(1E6, 8E5, 2E5)
4.0
```

fluids.**size_control_valve_l**(*rho*, *Psat*, *Pc*, *mu*, *P1*, *P2*, *Q*, *D1*, *D2*, *d*, *FL*, *Fd*)

Calculates flow coefficient of a control valve passing a liquid according to IEC 60534. Uses a large number of inputs in SI units. Note the return value is not standard SI. All parameters are required. This sizing model does not officially apply to liquid mixtures, slurries, non-Newtonian fluids, or liquid-solid conveyance systems. For details of the calculations, consult *[R9]*.

Parameters **rho** : float

Density of the liquid at the inlet [kg/m^3]

Psat : float

Saturation pressure of the fluid at inlet temperature [Pa]

Pc : float

Critical pressure of the fluid [Pa]

mu : float

Viscosity of the fluid [Pa*s]

P1 : float

Inlet pressure of the fluid before valves and reducers [Pa]

P2 : float

Outlet pressure of the fluid after valves and reducers [Pa]

Q : float

Volumetric flow rate of the fluid [m^3/s]

D1 : float

Diameter of the pipe before the valve [m]

D2 : float

Diameter of the pipe after the valve [m]

d : float

Diameter of the valve [m]

FL : float

Liquid pressure recovery factor of a control valve without attached fittings []

Fd : float

Valve style modifier []

Returns **C** : float

Kv flow coefficient [m^3/hr at a dP of 1 bar]

References

[R9]

Examples

From *[R9]*, matching example 1 for a globe, parabolic plug, flow-to-open valve.

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
... P1=680E3, P2=220E3, Q=0.1, D1=0.15, D2=0.15, d=0.15,
... FL=0.9, Fd=0.46)
164.9954763704956
```

From *[R9]*, matching example 2 for a ball, segmented ball, flow-to-open valve.

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
... P1=680E3, P2=220E3, Q=0.1, D1=0.1, D2=0.1, d=0.1,
... FL=0.6, Fd=0.98)
238.05817216710483
```

Modified example 1 with non-choked flow, with reducer and expander

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
...  P1=680E3, P2=220E3, Q=0.1, D1=0.1, D2=0.09, d=0.08, FL=0.9, Fd=0.46)
177.44417090966715
```

Modified example 2 with non-choked flow, with reducer and expander

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-4,
... P1=680E3, P2=220E3, Q=0.1, D1=0.1, D2=0.1, d=0.95, FL=0.6, Fd=0.98)
230.1734424266345
```

Modified example 2 with laminar flow at 100x viscosity, 100th flow rate, and 1/10th diameters:

```
>>> size_control_valve_l(rho=965.4, Psat=70.1E3, Pc=22120E3, mu=3.1472E-2,
... P1=680E3, P2=220E3, Q=0.001, D1=0.01, D2=0.01, d=0.01, FL=0.6, Fd=0.98)
3.0947562381723626
```

fluids.**size_control_valve_g**(*T*, *MW*, *mu*, *gamma*, *Z*, *P1*, *P2*, *Q*, *D1*, *D2*, *d*, *FL*, *Fd*, *xT*)

Calculates flow coefficient of a control valve passing a gas according to IEC 60534. Uses a large number of inputs in SI units. Note the return value is not standard SI. All parameters are required. For details of the calculations, consult *[R10]*. Note the inlet gas flow conditions.

Parameters **T** : float

Temperature of the gas at the inlet [K]

MW : float

Molecular weight of the gas [g/mol]

mu : float

Viscosity of the fluid at inlet conditions [Pa*s]

gamma : float

Specific heat capacity ratio [-]

Z : float

Compressibility factor at inlet conditions, [-]

P1 : float

Inlet pressure of the gas before valves and reducers [Pa]

P2 : float

Outlet pressure of the gas after valves and reducers [Pa]

Q : float

Volumetric flow rate of the gas at *273.15 K* and 1 atm specifically [m^3/s]

D1 : float

Diameter of the pipe before the valve [m]

D2 : float

Diameter of the pipe after the valve [m]

d : float

Diameter of the valve [m]

FL : float

Liquid pressure recovery factor of a control valve without attached fittings []

Fd : float

Valve style modifier []

xT : float

Pressure difference ratio factor of a valve without fittings at choked flow [-]

Returns **C** : float

Kv flow coefficient [m^3/hr at a dP of 1 bar]

References

[R10]

Examples

From *[R10]*, matching example 3 for non-choked gas flow with attached fittings and a rotary, eccentric plug, flow-to-open control valve:

```
>>> size_control_valve_g(T=433., MW=44.01, mu=1.4665E-4, gamma=1.30,
... Z=0.988, P1=680E3, P2=310E3, Q=38/36., D1=0.08, D2=0.1, d=0.05,
... FL=0.85, Fd=0.42, xT=0.60)
72.58664545391052
```

From *[R10]*, roughly matching example 4 for a small flow trim sized tapered needle plug valve. Difference is 3% and explained by the difference in algorithms used.

```
>>> size_control_valve_g(T=320., MW=39.95, mu=5.625E-5, gamma=1.67, Z=1.0,
... P1=2.8E5, P2=1.3E5, Q=0.46/3600., D1=0.015, D2=0.015, d=0.015, FL=0.98,
... Fd=0.07, xT=0.8)
0.016498765335995726
```

fluids.**Reynolds**(*V*, *D*, *rho=None*, *mu=None*, *nu=None*)

Calculates Reynolds number or *Re* for a fluid with the given properties for the specified velocity and diameter.

$$Re = D \cdot V \nu = \frac{\rho V D}{\mu}$$

Inputs either of any of the following sets:

- V, D, density *rho* and kinematic viscosity *mu*
- V, D, and dynamic viscosity *nu*

Parameters **D** : float

Diameter [m]

V : float

Velocity [m/s]

rho : float, optional

Density, [kg/m^3]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

Returns **Re** : float

Reynolds number []

Notes

$$Re = \frac{\text{Momentum}}{\text{Viscosity}}$$

An error is raised if none of the required input sets are provided.

References

[R11], *[R12]*

Examples

```
>>> Reynolds(2.5, 0.25, 1.1613, 1.9E-5)
38200.65789473684
>>> Reynolds(2.5, 0.25, nu=1.636e-05)
38202.93398533008
```

fluids.**Prandtl**(*Cp=None*, *k=None*, *mu=None*, *nu=None*, *rho=None*, *alpha=None*)

Calculates Prandtl number or *Pr* for a fluid with the given parameters.

$$Pr = \frac{C_p \mu}{k} = \frac{\nu}{\alpha} = \frac{C_p \rho \nu}{k}$$

Inputs can be any of the following sets:

- Heat capacity, dynamic viscosity, and thermal conductivity
- Thermal diffusivity and kinematic viscosity
- Heat capacity, kinematic viscosity, thermal conductivity, and density

Parameters **Cp** : float

Heat capacity, [J/kg/K]

k : float

Thermal conductivity, [W/m/K]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

rho : float

Density, [kg/m^3]

alpha : float

Thermal diffusivity, [m^2/s]

Returns **Pr** : float

Prandtl number []

Notes

$$Pr = \frac{\text{kinematic viscosity}}{\text{thermal diffusivity}} = \frac{\text{momentum diffusivity}}{\text{thermal diffusivity}}$$

An error is raised if none of the required input sets are provided.

References

[R13], *[R14]*, *[R15]*

Examples

```
>>> Prandtl(Cp=1637., k=0.010, mu=4.61E-6)
0.754657
>>> Prandtl(Cp=1637., k=0.010, nu=6.4E-7, rho=7.1)
0.7438528
>>> Prandtl(nu=6.3E-7, alpha=9E-7)
0.7000000000000001
```

`fluids.`**`Grashof`**(*L*, *beta*, *T1*, *T2=0*, *rho=None*, *mu=None*, *nu=None*, *g=9.80665*)

Calculates Grashof number or *Gr* for a fluid with the given properties, temperature difference, and characteristic length.

$$Gr = \frac{g\beta(T_s - T_\infty)L^3}{\nu^2} = \frac{g\beta(T_s - T_\infty)L^3\rho^2}{\mu^2}$$

Inputs either of any of the following sets:

- L, beta, T1 and T2, and density *rho* and kinematic viscosity *mu*
- L, beta, T1 and T2, and dynamic viscosity *nu*

Parameters **L** : float

Characteristic length [m]

beta : float

Volumetric thermal expansion coefficient [1/K]

T1 : float

Temperature 1, usually a film temperature [K]

T2 : float, optional

Temperature 2, usually a bulk temperature (or 0 if only a difference is provided to the function) [K]

rho : float, optional

Density, [kg/m^3]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

g : float, optional

Acceleration due to gravity, [m/s^2]

Returns **Gr** : float

Grashof number []

Notes

$$Gr = \frac{\text{Buoyancy forces}}{\text{Viscous forces}}$$

An error is raised if none of the required input sets are provided. Used in free convection problems only.

References

[R16], *[R17]*

Examples

Example 4 of *[R16]*, p. 1-21 (matches):

```
>>> Grashof(L=0.9144, beta=0.000933, T1=178.2, rho=1.1613, mu=1.9E-5)
4656936556.178915
>>> Grashof(L=0.9144, beta=0.000933, T1=378.2, T2=200, nu=1.636e-05)
4657491516.530312
```

`fluids.`**`Nusselt`**(*h*, *L*, *k*)

Calculates Nusselt number *Nu* for a heat transfer coefficient *h*, characteristic length *L*, and thermal conductivity *k*.

$$Nu = \frac{hL}{k}$$

Parameters **h** : float

Heat transfer coefficient, [W/m^2/K]

L : float

Characteristic length, no typical definition [m]

k : float

Thermal conductivity of fluid [W/m/K]

Returns **Nu** : float

Nusselt number, [-]

Notes

Do not confuse k, the thermal conductivity of the fluid, with that of within a solid object associated with!

$$Nu = \frac{\text{Convective heat transfer}}{\text{Conductive heat transfer}}$$

References

[R18], *[R19]*

Examples

```
>>> Nusselt(1000., 1.2, 300.)
4.0
>>> Nusselt(10000., .01, 4000.)
0.025
```

fluids.**Sherwood**(*K*, *L*, *D*)

Calculates Sherwood number *Sh* for a mass transfer coefficient *K*, characteristic length *L*, and diffusivity *D*.

$$Sh = \frac{KL}{D}$$

Parameters **K** : float

Mass transfer coefficient, [m/s]

L : float

Characteristic length, no typical definition [m]

D : float

Diffusivity of a species [m/s^2]

Returns **Sh** : float

Sherwood number, [-]

Notes

$$Sh = \frac{\text{Mass transfer by convection}}{\text{Mass transfer by diffusion}} = \frac{K}{D/L}$$

References

[R20]

Examples

```
>>> Sherwood(1000., 1.2, 300.)
4.0
```

fluids.**Rayleigh**(*Pr*, *Gr*)

Calculates Rayleigh number or *Ra* using Prandtl number *Pr* and Grashof number *Gr* for a fluid with the given properties, temperature difference, and characteristic length used to calculate *Gr* and *Pr*.

$$Ra = PrGr$$

Parameters **Pr** : float

Prandtl number []

Gr : float

Grashof number []

Returns **Ra** : float

Rayleigh number []

Notes

Used in free convection problems only.

References

[R21], *[R22]*

Examples

```
>>> Rayleigh(1.2, 4.6E9)
5520000000.0
```

fluids.**Schmidt**(*D*, *mu=None*, *nu=None*, *rho=None*)

Calculates Schmidt number or *Sc* for a fluid with the given parameters.

$$Sc = \frac{\mu}{D\rho} = \frac{\nu}{D}$$

Inputs can be any of the following sets:

- Diffusivity, dynamic viscosity, and density
- Diffusivity and kinematic viscosity

Parameters **D** : float

Diffusivity of a species, [m^2/s]

mu : float, optional

Dynamic viscosity, [Pa*s]

nu : float, optional

Kinematic viscosity, [m^2/s]

rho : float, optional

Density, [kg/m^3]

Returns **Sc** : float

Schmidt number []

Notes

$$Sc = \frac{\text{kinematic viscosity}}{\text{molecular diffusivity}} = \frac{\text{viscous diffusivity}}{\text{species diffusivity}}$$

An error is raised if none of the required input sets are provided.

References

[R23], *[R24]*

Examples

```
>>> Schmidt(D=2E-6, mu=4.61E-6, rho=800)
0.00288125
>>> Schmidt(D=1E-9, nu=6E-7)
599.9999999999999
```

`fluids.`**`Peclet_heat`**(*V*, *L*, *rho=None*, *Cp=None*, *k=None*, *alpha=None*)

Calculates heat transfer Peclet number or *Pe* for a specified velocity *V*, characteristic length *L*, and specified properties for the given fluid.

$$Pe = \frac{VL\rho C_p}{k} = \frac{LV}{\alpha}$$

Inputs either of any of the following sets:

- V, L, density *rho*, heat capcity *Cp*, and thermal conductivity *k*
- V, L, and thermal diffusivity *alpha*

Parameters **V** : float

Velocity [m/s]

L : float

Characteristic length [m]

rho : float, optional

Density, [kg/m^3]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Returns **Pe** : float

Peclet number (heat) []

Notes

$$Pe = \frac{\text{Bulk heat transfer}}{\text{Conduction heat transfer}}$$

An error is raised if none of the required input sets are provided.

References

[R25], *[R26]*

Examples

```
>>> Peclet_heat(1.5, 2, 1000., 4000., 0.6)
20000000.0
>>> Peclet_heat(1.5, 2, alpha=1E-7)
30000000.0
```

fluids.**Peclet_mass**(*V*, *L*, *D*)

Calculates mass transfer Peclet number or *Pe* for a specified velocity *V*, characteristic length *L*, and diffusion coefficient *D*.

$$Pe = \frac{LV}{D}$$

Parameters **V** : float

Velocity [m/s]

L : float

Characteristic length [m]

D : float

Diffusivity of a species, [m^2/s]

Returns **Pe** : float

Peclet number (mass) []

Notes

$$Pe = \frac{\text{Advective transport rate}}{\text{Diffusive transport rate}}$$

References

[R27]

Examples

```
>>> Peclet_mass(1.5, 2, 1E-9)
3000000000.0
```

fluids.**Fourier_heat**(*t*, *L*, *rho=None*, *Cp=None*, *k=None*, *alpha=None*)

Calculates heat transfer Fourier number or *Fo* for a specified time *t*, characteristic length *L*, and specified properties for the given fluid.

$$Fo = \frac{kt}{C_p \rho L^2} = \frac{\alpha t}{L^2}$$

Inputs either of any of the following sets:

- t, L, density *rho*, heat capcity *Cp*, and thermal conductivity *k*
- t, L, and thermal diffusivity *alpha*

Parameters **t** : float

time [s]

L : float

Characteristic length [m]

rho : float, optional

Density, [kg/m^3]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Returns **Fo** : float

Fourier number (heat) []

Notes

$$Fo = \frac{\text{Heat conduction rate}}{\text{Rate of thermal energy storage in a solid}}$$

An error is raised if none of the required input sets are provided.

References

[R28], *[R29]*

Examples

```
>>> Fourier_heat(1.5, 2, 1000., 4000., 0.6)
5.625e-08
>>> Fourier_heat(1.5, 2, alpha=1E-7)
3.75e-08
```

fluids.**Fourier_mass**(*t*, *L*, *D*)

Calculates mass transfer Fourier number or *Fo* for a specified time *t*, characteristic length *L*, and diffusion coefficient *D*.

$$Fo = \frac{Dt}{L^2}$$

Parameters **t** : float

time [s]

L : float

Characteristic length [m]

D : float

Diffusivity of a species, [m^2/s]

Returns **Fo** : float

Fourier number (mass) []

Notes

$$Fo = \frac{\text{Diffusive transport rate}}{\text{Storage rate}}$$

References

[R30]

Examples

```
>>> Fourier_mass(1.5, 2, 1E-9)
3.7500000000000005e-10
```

fluids.**Graetz_heat**(*V*, *D*, *x*, *rho=None*, *Cp=None*, *k=None*, *alpha=None*)

Calculates Graetz number or *Gz* for a specified velocity *V*, diameter *D*, axial diatance *x*, and specified properties for the given fluid.

$$Gz = \frac{VD^2 \cdot C_p\rho}{x \cdot k} = \frac{VD^2}{x\alpha}$$

Inputs either of any of the following sets:

•V, D, x, density *rho*, heat capcity *Cp*, and thermal conductivity *k*

•V, D, x, and thermal diffusivity *alpha*

Parameters **V** : float

Velocity, [m/s]

D : float

Diameter [m]

x : float

Axial distance [m]

rho : float, optional

Density, [kg/m^3]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Returns **Gz** : float

Graetz number []

Notes

$$Gz = \frac{\text{Time for radial heat diffusion in a fluid by conduction}}{\text{Time taken by fluid to reach distance x}}$$

$$Gz = \frac{D}{x} RePr$$

An error is raised if none of the required input sets are provided.

References

[R31]

Examples

```
>>> Graetz_heat(1.5, 0.25, 5, 800., 2200., 0.6)
55000.0
>>> Graetz_heat(1.5, 0.25, 5, alpha=1E-7)
187500.0
```

fluids.**Lewis**(*D=None*, *alpha=None*, *Cp=None*, *k=None*, *rho=None*)

Calculates Lewis number or *Le* for a fluid with the given parameters.

$$Le = \frac{k}{\rho C_p D} = \frac{\alpha}{D}$$

Inputs can be either of the following sets:

- Diffusivity and Thermal diffusivity
- Diffusivity, heat capacity, thermal conductivity, and density

Parameters **D** : float

Diffusivity of a species, [m^2/s]

alpha : float, optional

Thermal diffusivity, [m^2/s]

Cp : float, optional

Heat capacity, [J/kg/K]

k : float, optional

Thermal conductivity, [W/m/K]

rho : float, optional

Density, [kg/m^3]

Returns **Le** : float

Lewis number []

Notes

$$Le = \frac{\text{Thermal diffusivity}}{\text{Mass diffusivity}} = \frac{Sc}{Pr}$$

An error is raised if none of the required input sets are provided.

References

[R32], *[R33]*, *[R34]*

Examples

```
>>> Lewis(D=22.6E-6, alpha=19.1E-6)
0.8451327433628318
>>> Lewis(D=22.6E-6, rho=800., k=.2, Cp=2200)
0.00502815768302494
```

fluids.**Weber**(*V*, *L*, *rho*, *sigma*)

Calculates Weber number, *We*, for a fluid with the given density, surface tension, velocity, and geometric parameter (usually diameter of bubble).

$$We = \frac{V^2 L \rho}{\sigma}$$

Parameters **V** : float

Velocity of fluid, [m/s]

L : float

Characteristic length, typically bubble diameter [m]

rho : float

Density of fluid, [kg/m^3]

sigma : float

Surface tension, [N/m]

Returns **We** : float

Weber number []

Notes

Used in bubble calculations.

$$We = \frac{\text{inertial force}}{\text{surface tension force}}$$

References

[R35], *[R36]*, *[R37]*

Examples

```
>>> Weber(V=0.18, L=0.001, rho=900., sigma=0.01)
2.916
```

fluids.**Mach**(*V*, *c*)

Calculates Mach number or *Ma* for a fluid of velocity *V* with speed of sound *c*.

$$Ma = \frac{V}{c}$$

Parameters **V** : float

Velocity of fluid, [m/s]

c : float

Speed of sound in fluid, [m/s]

Returns **Ma** : float

Mach number []

Notes

Used in compressible flow calculations.

$$Ma = \frac{\text{fluid velocity}}{\text{sonic velocity}}$$

References

[R38], *[R39]*

Examples

```
>>> Mach(33., 330)
0.1
```

`fluids.`**`Knudsen`**(*path*, *L*)

Calculates Knudsen number or *Kn* for a fluid with mean free path *path* and for a characteristic length *L*.

$$Kn = \frac{\lambda}{L}$$

Parameters **path** : float

Mean free path between molecular collisions, [m]

L : float

Characteristic length, [m]

Returns **Kn** : float

Knudsen number []

Notes

Used in mass transfer calculations.

$$Kn = \frac{\text{Mean free path length}}{\text{Characteristic length}}$$

References

[R40], *[R41]*

Examples

```
>>> Knudsen(1e-10, .001)
1e-07
```

`fluids.`**`Bond`**(*rhol*, *rhog*, *sigma*, *L*)

Calculates Bond number, *Bo* also known as Eotvos number, for a fluid with the given liquid and gas densities, surface tension, and geometric parameter (usually length).

$$Bo = \frac{g(\rho_l - \rho_g)L^2}{\sigma}$$

Parameters **rhol** : float

Density of liquid, [kg/m^3]

rhog : float

Density of gas, [kg/m^3]

sigma : float

Surface tension, [N/m]

L : float

Characteristic length, [m]

Returns **Bo** : float

Bond number []

References

[R42]

Examples

```
>>> Bond(1000., 1.2, .0589, 2)
665187.2339558573
```

fluids.**Froude**(*V*, *L*, *g=9.80665*, *squared=False*)

Calculates Froude number *Fr* for velocity *V* and geometric length *L*. If desired, gravity can be specified as well. Normally the function returns the result of the equation below; Froude number is also often said to be defined as the square of the equation below.

$$Fr = \frac{V}{\sqrt{gL}}$$

Parameters **V** : float

Velocity of the particle or fluid, [m/s]

L : float

Characteristic length, no typical definition [m]

g : float, optional

Acceleration due to gravity, [m/s^2]

squared : bool, optional

Whether to return the squared form of Frounde number

Returns **Fr** : float

Froude number, [-]

Notes

Many alternate definitions including density ratios have been used.

$$Fr = \frac{\text{Inertial Force}}{\text{Gravity Force}}$$

References

[R43], *[R44]*

Examples

```
>>> Froude(1.83, L=2., g=1.63)
1.0135432593877318
>>> Froude(1.83, L=2., squared=True)
0.17074638128208924
```

fluids.**Strouhal**(*f*, *L*, *V*)

Calculates Strouhal number *St* for a characteristic frequency *f*, characteristic length *L*, and velocity *V*.

$$St = \frac{fL}{V}$$

Parameters **f** : float

Characteristic frequency, usually that of vortex shedding, [Hz]

L : float

Characteristic length, [m]

V : float

Velocity of the fluid, [m/s]

Returns **St** : float

Strouhal number, [-]

Notes

Sometimes abbreviated to S or Sr.

$$St = \frac{\text{Characteristif flow time}}{\text{Period of oscillation}}$$

References

[R45], *[R46]*

Examples

```
    >>> Strouhal(8, 2., 4.)
    4.0
```

fluids.**Biot**(*h*, *L*, *k*)

Calculates Biot number *Br* for heat transfer coefficient *h*, geometric length *L*, and thermal conductivity *k*.

$$Bi = \frac{hL}{k}$$

Parameters **h** : float

Heat transfer coefficient, [W/m^2/K]

L : float

Characteristic length, no typical definition [m]

k : float

Thermal conductivity, within the object [W/m/K]

Returns **Bi** : float

Biot number, [-]

Notes

Do not confuse k, the thermal conductivity within the object, with that of the medium h is calculated with!

$$Bi = \frac{\text{Surface thermal resistance}}{\text{Internal thermal resistance}}$$

References

[R47], *[R48]*

Examples

```
    >>> Biot(1000., 1.2, 300.)
    4.0
    >>> Biot(10000., .01, 4000.)
    0.025
```

fluids.**Stanton**(*h*, *V*, *rho*, *Cp*)

Calculates Stanton number or *St* for a specified heat transfer coefficient *h*, velocity *V*, density *rho*, and heat capacity *Cp*.

$$St = \frac{h}{V\rho Cp}$$

Parameters **h** : float

Heat transfer coefficient, [W/m^2/K]

V : float

Velocity, [m/s]

rho : float

Density, [kg/m^3]

Cp : float

Heat capacity, [J/kg/K]

Returns **St** : float

Stanton number []

Notes

$$St = \frac{\text{Heat transfer coefficient}}{\text{Thermal capacity}}$$

References

[R49], *[R49]*

Examples

```
>>> Stanton(5000, 5, 800, 2000.)
0.000625
```

fluids.**Euler**(*dP*, *rho*, *V*)

Calculates Euler number or *Eu* for a fluid of velocity *V* and density *rho* experiencing a pressure drop *dP*.

$$Eu = \frac{\Delta P}{\rho V^2}$$

Parameters **dP** : float

Pressure drop experience by the fluid, [Pa]

rho : float

Density of the fluid, [kg/m^3]

V : float

Velocity of fluid, [m/s]

Returns **Eu** : float

Euler number []

Notes

Used in pressure drop calculations. Rarely, this number is divided by two. Named after Leonhard Euler applied calculus to fluid dynamics.

$$Eu = \frac{\text{Pressure drop}}{2 \cdot \text{velocity head}}$$

References

[R51], *[R52]*

Examples

```
>>> Euler(1E5, 1000., 4)
6.25
```

fluids.**Cavitation**(*P*, *Psat*, *rho*, *V*)

Calculates Cavitation number or *Ca* for a fluid of velocity *V* with a pressure *P*, vapor pressure *Psat*, and density *rho*.

$$Ca = \sigma_c = \sigma = \frac{P - P_{sat}}{\frac{1}{2}\rho V^2}$$

Parameters **P** : float

Internal pressure of the fluid, [Pa]

Psat : float

Vapor pressure of the fluid, [Pa]

rho : float

Density of the fluid, [kg/m^3]

V : float

Velocity of fluid, [m/s]

Returns **Ca** : float

Cavitation number []

Notes

Used in determining if a flow through a restriction will cavitate. Sometimes, the multiplication by 2 will be omitted;

$$Ca = \frac{\text{Pressure - Vapor pressure}}{\text{Inertial pressure}}$$

References

[R53], *[R54]*

Examples

```
>>> Cavitation(2E5, 1E4, 1000, 10)
3.8
```

fluids.**Eckert**(*V*, *Cp*, *dT*)

Calculates Eckert number or *Ec* for a fluid of velocity *V* with a heat capacity *Cp*, between two temperature given as *dT*.

$$Ec = \frac{V^2}{C_p \Delta T}$$

Parameters **V** : float

Velocity of fluid, [m/s]

Cp : float

Heat capacity of the fluid, [J/kg/K]

dT : float

Temperature difference, [K]

Returns **Ec** : float

Eckert number []

Notes

Used in certain heat transfer calculations. Fairly rare.

$$Ec = \frac{\text{Kinetic energy}}{\text{Enthalpy difference}}$$

References

[R55]

Examples

```
>>> Eckert(10, 2000., 25.)
0.002
```

fluids.**Jakob**(*Cp*, *Hvap*, *Te*)

Calculates Jakob number or *Ja* for a boiling fluid with sensible heat capacity *Cp*, enthalpy of vaporization *Hvap*, and boiling at *Te* degrees above its saturation boiling point.

$$Ja = \frac{C_P \Delta T_e}{\Delta H_{vap}}$$

Parameters **Cp** : float

Heat capacity of the fluid, [J/kg/K]

Hvap : float

Enthalpy of vaporization of the fluid at its saturation temperature [J/kg]

Te : float

Temperature difference above the fluid's saturation boiling temperature, [K]

Returns **Ja** : float

Jakob number []

Notes

Used in boiling heat transfer analysis. Fairly rare.

$$Ja = \frac{\Delta\text{Sensible heat}}{\Delta\text{Latent heat}}$$

References

[R56], *[R57]*

Examples

```
>>> Jakob(4000., 2E6, 10.)
0.02
```

fluids.**Power_number**(*P*, *L*, *N*, *rho*)

Calculates power number, *Po*, for an agitator applying a specified power *P* with a characteristic length *L*, rotationa speed *N*, to a fluid with a specified density *rho*.

$$Po = \frac{P}{\rho N^3 D^5}$$

Parameters **P** : float

Power applied, [W]

L : float

Characteristic length, typically agitator diameter [m]

N : float

Speed [revolutions/second]

rho : float

Density of fluid, [kg/m^3]

Returns **Po** : float

Power number []

Notes

Used in mixing calculations.

$$Po = \frac{\text{Power}}{\text{Rotational inertia}}$$

References

[R58], *[R59]*

Examples

```
>>> Power_number(P=180, L=0.01, N=2.5, rho=800.)
144000000.0
```

fluids.**Drag**(*F*, *A*, *V*, *rho*)

Calculates drag coefficient *Cd* for a given drag force *F*, projected area *A*, characteristic velocity *V*, and density *rho*.

$$C_D = \frac{F_d}{A \cdot \frac{1}{2}\rho V^2}$$

Parameters **F** : float

Drag force, [N]

A : float

Projected area, [m^2]

V : float

Characteristic velocity, [m/s]

rho : float

Density, [kg/m^3]

Returns **Cd** : float

Drag coefficient, [-]

Notes

Used in flow around objects, or objects flowing within a fluid.

$$C_D = \frac{\text{Drag forces}}{\text{Projected area} \cdot \text{Velocity head}}$$

References

[R60], *[R61]*

Examples

```
>>> Drag(1000, 0.0001, 5, 2000)
400.0
```

fluids.**Capillary**(*V*, *mu*, *sigma*)

Calculates Capillary number *Ca* for a characteristic velocity *V*, viscosity *mu*, and surface tension *sigma*.

$$Ca = \frac{V\mu}{\sigma}$$

Parameters **V** : float

Characteristic velocity, [m/s]

mu : float

Dynamic viscosity, [Pa*s]

sigma : float

Surface tension, [N/m]

Returns **Ca** : float

Capillary number, [-]

Notes

Used in porous media calculations and film flow calculations. Surface tension may gas-liquid, or liquid-liquid.

$$Ca = \frac{\text{Viscous forces}}{\text{Surface forces}}$$

References

[R62], *[R63]*

Examples

```
>>> Capillary(1.2, 0.01, .1)
0.12
```

fluids.**Archimedes**(*L*, *rhof*, *rhop*, *mu*, *g=9.80665*)

Calculates Archimedes number, *Ar*, for a fluid and particle with the given densities, characteristic length, viscosity, and gravity (usually diameter of particle).

$$Ar = \frac{L^3 \rho_f (\rho_p - \rho_f) g}{\mu^2}$$

Parameters **L** : float

Characteristic length, typically particle diameter [m]

rhof : float

Density of fluid, [kg/m^3]

rhop : float

Density of particle, [kg/m^3]

mu : float

Viscosity of fluid, [N/m]

g : float, optional

Acceleration due to gravity, [m/s^2]

Returns **Ar** : float

Archimedes number []

Notes

Used in fluid-particle interaction calculations.

$$Ar = \frac{\text{Gravitational force}}{\text{Viscous force}}$$

References

[R64], *[R65]*

Examples

```
>>> Archimedes(0.002, 0.2804, 2699.37, 4E-5)
37109.575890227665
```

fluids.**Ohnesorge**(*L*, *rho*, *mu*, *sigma*)

Calculates Ohnesorge number, *Oh*, for a fluid with the given characteristic length, density, viscosity, and surface tension.

$$\text{Oh} = \frac{\mu}{\sqrt{\rho \sigma L}}$$

Parameters **L** : float

Characteristic length [m]

rho : float

Density of fluid, [kg/m^3]

mu : float

Viscosity of fluid, [Pa*s]

sigma : float

Surface tension, [N/m]

Returns **Oh** : float

Ohnesorge number []

Notes

Often used in spray calculations. Sometimes given the symbol Z.

$$Oh = \frac{\sqrt{\text{We}}}{\text{Re}} = \frac{\text{viscous forces}}{\sqrt{\text{Inertia} \cdot \text{Surface tension}}}$$

References

[R66]

Examples

```
>>> Ohnesorge(1E-5, 2000., 1E-4, 1E-1)
0.00223606797749979
```

fluids.**thermal_diffusivity**(*k*, *rho*, *Cp*)

Calculates thermal diffusivity or *alpha* for a fluid with the given parameters.

$$\alpha = \frac{k}{\rho Cp}$$

Parameters **k** : float

Thermal conductivity, [W/m/K]

Cp : float

Heat capacity, [J/kg/K]

rho : float

Density, [kg/m^3]

Returns **alpha** : float

Thermal diffusivity, [m^2/s]

References

[R67]

Examples

```
>>> thermal_diffusivity(0.02, 1., 1000.)
2e-05
```

fluids.**c_ideal_gas**(*T*, *k*, *MW*)

Calculates speed of sound *c* in an ideal gas at tempereture T.

$$c = \sqrt{kR_{specific}T}$$

Parameters **T** : float

Temperature of fluid, [K]

k : float

Isentropic exponent of fluid, [-]

MW : float

Molecular weight of fluid, [g/mol]

Returns **c** : float

Speed of sound in fluid, [m/s]

Notes

Used in compressible flow calculations. Note that the gas constant used is the specific gas constant:

$$R_{specific} = R\frac{1000}{MW}$$

References

[R68], *[R69]*

Examples

```
>>> c_ideal_gas(1.4, 303., 28.96)
348.9820361755092
```

fluids.**relative_roughness**(*D*, *roughness=1.52e-06*)

Calculates relative roughness *eD* using a diameter and the roughness of the material of the wall. Default roughness is that of steel.

$$eD = \frac{\epsilon}{D}$$

Parameters **D** : float

Diameter of pipe, [m]

roughness : float, optional

Roughness of pipe wall [m]

Returns **eD** : float

Relative Roughness, [-]

References

[R70], *[R71]*

Examples

```
>>> relative_roughness(0.0254)
5.9842519685039374e-05
>>> relative_roughness(0.5, 1E-4)
0.0002
```

fluids.**nu_mu_converter**(*rho*, *nu=0*, *mu=0*)

Specify density and either Kinematic viscosity or Dynamic Viscosity by name for the result to be converted to the other.

```
>>> nu_mu_converter(998.1,nu=1.01E-6)
0.001008081
```

fluids.**gravity**(*latitude*, *height*)

Calculates local acceleration due to gravity g according to *[R72]*. Uses latitude and height to calculate g.

$$g = 9.780356(1 + 0.0052885\sin^2\phi - 0.0000059^2 2\phi) - 3.086\times 10^{-6}H$$

Parameters **latitude** : float

Degrees, [degrees]

height : float

Height above earth's surface [m]

Returns **g** : float

Acceleration due to gravity, [m/s^2]

Notes

Better models, such as EGM2008 exist.

References

[R72]

Examples

```
>>> gravity(55, 1E4)
9.784151976863571
```

fluids.**K_from_f**(*f*, *L*, *D*)

Calculates loss coefficient, K, for a given section of pipe at a specified friction factor.

$$K = fL/D$$

Parameters **f** : float

friction factor of pipe, []

L : float

Length of pipe, [m]

D : float

Inner diameter of pipe, [m]

Returns **K** : float

Loss coefficient, []

Notes

For fittings with a specified L/D ratio, use D = 1 and set L to specified L/D ratio.

Examples

```
>>> K_from_f(f=0.018, L=100., D=.3)
6.0
```

fluids.**K_from_L_equiv**(*L_D*, *f=0.015*)

Calculates loss coefficient, for a given equivalent length (L/D).

$$K = f\frac{L}{D}$$

Parameters **L_D** : float

Length over diameter, []

f : float, optional

Darcy friction factor, [-]

Returns **K** : float

Loss coefficient, []

Notes

Almost identical to *K_from_f*, but with a default friction factor for fully turbulent flow in steel pipes.

Examples

```
>>> K_from_L_equiv(240.)
3.5999999999999996
```

fluids.**dP_from_K**(*K*, *rho*, *V*)

Calculates pressure drop, for a given loss coefficient, at a specified density and velocity.

$$dP = 0.5K\rho V^2$$

Parameters **K** : float

Loss coefficient, []

rho : float

Density of fluid, [kg/m^3]

V : float

Velocity of fluid in pipe, [m/s]

Returns **dP** : float

Pressure drop, [Pa]

Notes

Loss ciefficient *K* is usually the sum of several factors, including the friction factor.

Examples

```
>>> dP_from_K(K=10, rho=1000, V=3)
45000.0
```

fluids.**head_from_K**(*K*, *V*)

Calculates head loss, for a given loss coefficient, at a specified velocity.

$$\text{head} = \frac{KV^2}{2g}$$

Parameters **K** : float

Loss coefficient, []

V : float

Velocity of fluid in pipe, [m/s]

Returns **head** : float

Head loss, [m]

Notes

Loss ciefficient *K* is usually the sum of several factors, including the friction factor.

Examples

```
>>> head_from_K(K=10, V=1.5)
1.1471807396001694
```

fluids.**head_from_P**(*P*, *rho*)

Calculates head for a fluid of specified density at specified pressure.

$$\text{head} = \frac{P}{\rho g}$$

Parameters **P** : float

Pressure fluid in pipe, [Pa]

rho : float

Density of fluid, [kg/m^3]

Returns **head** : float

Head, [m]

Notes

By definition. Head varies with location, inversely propertional to the increase in gravitational constant.

Examples

```
>>> head_from_P(P=1E5, rho=1000.)
10.197162129779283
```

fluids.**P_from_head**(*head*, *rho*)

Calculates head for a fluid of specified density at specified pressure.

$$P = \rho g \cdot \text{head}$$

Parameters **head** : float

Head, [m]

rho : float

Density of fluid, [kg/m^3]

Returns **P** : float

Pressure fluid in pipe, [Pa]

Examples

```
>>> P_from_head(head=5., rho=800.)
39226.6
```

fluids.**round_edge_screen**(*alpha*, *Re*, *angle=0*)

Returns the loss coefficient for a round edged wire screen or bar screen, as shown in *[R73]*. Angle of inclination may be specified as well.

Parameters **alpha** : float

Fraction of screen open to flow [-]

Re : float

Reynolds number of flow through screen with D = space between rods, []

angle : float, optional

Angle of inclination, with 0 being straight and 90 being parallel to flow [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

Linear interpolation between a table of values. Re should be > 20. alpha should be between 0.05 and 0.8. Angle may be no more than 85 degress.

References

[R73]

Examples

```
>>> round_edge_screen(0.5, 100)
2.0999999999999996
>>> round_edge_screen(0.5, 100, 45)
1.05
>>> round_edge_screen(0.5, 100, 85)
0.18899999999999997
```

fluids.**round_edge_open_mesh**(*alpha*, *subtype='diamond pattern wire'*, *angle=0*)

Returns the loss coefficient for a round edged open net/screen made of one of the following patterns, according to *[R74]*:

'round bar screen':

$$K = 0.95(1-\alpha) + 0.2(1-\alpha)^2$$

'diamond pattern wire':

$$K = 0.67(1-\alpha) + 1.3(1-\alpha)^2$$

'knotted net':

$$K = 0.70(1-\alpha) + 4.9(1-\alpha)^2$$

'knotless net':

$$K = 0.72(1-\alpha) + 2.1(1-\alpha)^2$$

Parameters **alpha** : float

Fraction of net/screen open to flow [-]

subtype : str

One of 'round bar screen', 'diamond pattern wire', 'knotted net' or 'knotless net'.

angle : float, optional

Angle of inclination, with 0 being straight and 90 being parallel to flow [degrees]

Returns **K** : float

Loss coefficient [-]

Notes

alpha should be between 0.85 and 1 for these correlations. Flow should be turbulent, with Re > 500. Angle may be no more than 85 degress.

References

[R74]

Examples

```
>>> [round_edge_open_mesh(0.88, i) for i in ['round bar screen',
... 'diamond pattern wire', 'knotted net', 'knotless net']]
[0.11687999999999998, 0.09912, 0.15455999999999998, 0.11664]
>>> round_edge_open_mesh(0.96, angle=33.)
0.02031327712601458
```

fluids.**square_edge_screen**(*alpha*)

Returns the loss coefficient for a square wire screen or square bar screen or perforated plate with squared edges, as shown in *[R75]*.

Parameters **alpha** : float

Fraction of screen open to flow [-]

Returns **K** : float

Loss coefficient [-]

Notes

Linear interpolation between a table of values.

References

[R75]

Examples

```
>>> square_edge_screen(0.99)
0.008000000000000007
```

fluids.**square_edge_grill**(*alpha=None*, *l=None*, *Dh=None*, *fd=None*)

Returns the loss coefficient for a square grill or square bar screen or perforated plate with squared edges of thickness l, as shown in *[R76]*.

for Dh < l < 50D

$$K = \frac{0.5(1-\alpha) + (1-\alpha^2)}{\alpha^2}$$

else:

$$K = \frac{0.5(1-\alpha) + (1-\alpha^2) + fl/D}{\alpha^2}$$

Parameters **alpha** : float

Fraction of grill open to flow [-]

l : float

Thickness of the grill or plate [m]

Dh : float

Hydraulic diameter of gap in grill, [m]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

If l, Dh, or fd is not provided, the first expression is used instead. The alteration of the expression to include friction factor is there if the grill is long enough to have considerable friction along the surface of the grill.

References

[R76]

Examples

```
>>> square_edge_grill(.45)
5.296296296296296
>>> square_edge_grill(.45, l=.15, Dh=.002, fd=.0185)
12.148148148148147
```

fluids.**round_edge_grill**(*alpha=None*, *l=None*, *Dh=None*, *fd=None*)

Returns the loss coefficient for a rounded square grill or square bar screen or perforated plate with rounded edges of thickness l, as shown in *[R77]*.

for Dh < l < 50D

$$K = lookup(alpha)$$

else:

$$K = lookup(alpha) + \frac{fl}{\alpha^2 D}$$

Parameters **alpha** : float

Fraction of grill open to flow [-]

l : float, optional

Thickness of the grill or plate [m]

Dh : float, optional

Hydraulic diameter of gap in grill, [m]

fd : float, optional

Darcy friction factor [-]

Returns **K** : float

Loss coefficient [-]

Notes

If l, Dh, or fd is not provided, the first expression is used instead. The alteration of the expression to include friction factor is there if the grill is long enough to have considerable friction along the surface of the grill. alpha must be between 0.3 and 0.7.

References

[R77]

Examples

```
>>> round_edge_grill(.45)
0.8
>>> round_edge_grill(.45, l=.15, Dh=.002, fd=.0185)
2.1875
```

fluids.**Moody**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Moody (1947) as shown in *[R78]* and originally in *[R79]*.

$$f_f = 1.375 \times 10^{-3}\left[1 + \left(2 \times 10^4 \frac{\epsilon}{D} + \frac{10^6}{Re}\right)^{1/3}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is Re >= 4E3 and Re <= 1E8; eD >= 0 < 0.01.

References

[R78], *[R79]*

Examples

```
>>> Moody(1E5, 1E-4)
0.01809185666808665
```

fluids.**Alshul_1952**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Alshul (1952) as shown in *[R80]*.

$$f_d = 0.11\left(\frac{68}{Re} + \frac{\epsilon}{D}\right)^{0.25}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R80]

Examples

```
>>> Alshul_1952(1E5, 1E-4)
0.018382997825686878
```

fluids.**Wood_1966**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Wood (1966) *[R82]* as shown in *[R81]*.

$$f_d = 0.094(\frac{\epsilon}{D})^{0.225} + 0.53(\frac{\epsilon}{D}) + 88(\frac{\epsilon}{D})^{0.4} Re^{-A_1}$$

$$A_1 = 1.62(\frac{\epsilon}{D})^{0.134}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 5E7; 1E-5 <= eD <= 4E-2.

References

[R81], *[R82]*

Examples

```
>>> Wood_1966(1E5, 1E-4)
0.021587570560090762
```

fluids.**Churchill_1973**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Churchill (1973) *[R84]* as shown in *[R83]*

$$\frac{1}{\sqrt{f_d}} = -2\log\left[\frac{\epsilon}{3.7D} + (\frac{7}{Re})^{0.9}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R83], *[R84]*

Examples

```
>>> Churchill_1973(1E5, 1E-4)
0.01846708694482294
```

fluids.**Eck_1973**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Eck (1973) *[R86]* as shown in *[R85]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left[\frac{\epsilon}{3.715D} + \frac{15}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R85], *[R86]*

Examples

```
>>> Eck_1973(1E5, 1E-4)
0.01775666973488564
```

fluids.**Jain_1976**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Jain (1976) *[R88]* as shown in *[R87]*.

$$\frac{1}{\sqrt{f_f}} = 2.28 - 4\log\left[\frac{\epsilon}{D} + \left(\frac{29.843}{Re}\right)^{0.9}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 5E3 <= Re <= 1E7; 4E-5 <= eD <= 5E-2.

References

[R87], *[R88]*

Examples

```
>>> Jain_1976(1E5, 1E-4)
0.018436560312693327
```

fluids.**Swamee_Jain_1976**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Swamee and Jain (1976) *[R90]* as shown in *[R89]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\left(\frac{6.97}{Re}\right)^{0.9} + (\frac{\epsilon}{3.7D})\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 5E3 <= Re <= 1E8; 1E-6 <= eD <= 5E-2.

References

[R89], *[R90]*

Examples

```
>>> Swamee_Jain_1976(1E5, 1E-4)
0.018452424431901808
```

fluids.**Churchill_1977**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Churchill and (1977) *[R92]* as shown in *[R91]*.

$$f_f = 2\left[(\frac{8}{Re})^{12} + (A_2 + A_3)^{-1.5}\right]^{1/12}$$

$$A_2 = \left\{2.457\ln\left[(\frac{7}{Re})^{0.9} + 0.27\frac{\epsilon}{D}\right]\right\}^{16}$$

$$A_3 = \left(\frac{37530}{Re}\right)^{16}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R91], *[R92]*

Examples

```
>>> Churchill_1977(1E5, 1E-4)
0.018462624566280075
```

fluids.**Chen_1979**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Chen (1979) *[R94]* as shown in *[R93]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\frac{\epsilon}{3.7065D} - \frac{5.0452}{Re}\log A_4\right]$$

$$A_4 = \frac{(\epsilon/D)^{1.1098}}{2.8257} + \left(\frac{7.149}{Re}\right)^{0.8981}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 4E8; 1E-7 <= eD <= 5E-2.

References

[R93], *[R94]*

Examples

```
>>> Chen_1979(1E5, 1E-4)
0.018552817507472126
```

fluids.**Round_1980**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Round (1980) *[R96]* as shown in *[R95]*.

$$\frac{1}{\sqrt{f_f}} = -3.6 \log\left[\frac{Re}{0.135 Re \frac{\epsilon}{D} + 6.5}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 4E8; 0 <= eD <= 5E-2.

References

[R95], *[R96]*

Examples

```
>>> Round_1980(1E5, 1E-4)
0.01831475391244354
```

fluids.**Shacham_1980**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Shacham (1980) *[R98]* as shown in *[R97]*.

$$\frac{1}{\sqrt{f_f}} = -4 \log\left[\frac{\epsilon}{3.7D} - \frac{5.02}{Re} \log\left(\frac{\epsilon}{3.7D} + \frac{14.5}{Re}\right)\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 4E8

References

[R97], *[R98]*

Examples

```
>>> Shacham_1980(1E5, 1E-4)
0.01860641215097828
```

fluids.**Barr_1981**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Barr (1981) *[R100]* as shown in *[R99]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left\{\frac{\epsilon}{3.7D} + \frac{4.518\log(\frac{Re}{7})}{Re\left[1+\frac{Re^{0.52}}{29}\left(\frac{\epsilon}{D}\right)^{0.7}\right]}\right\}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R99], *[R100]*

Examples

```
>>> Barr_1981(1E5, 1E-4)
0.01849836032779929
```

fluids.**Zigrang_Sylvester_1**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Zigrang and Sylvester (1982) *[R102]* as shown in *[R101]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\frac{\epsilon}{3.7D} - \frac{5.02}{Re}\log A_5\right]$$
$$A_5 = \frac{\epsilon}{3.7D} + \frac{13}{Re}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 4E-5 <= eD <= 5E-2.

References

[R101], *[R102]*

Examples

```
>>> Zigrang_Sylvester_1(1E5, 1E-4)
0.018646892425980794
```

fluids.**Zigrang_Sylvester_2**(*Re*, *eD*)

Calculates Darcy friction factor using the second method in Zigrang and Sylvester (1982) *[R104]* as shown in *[R103]*.

$$\frac{1}{\sqrt{f_f}} = -4\log\left[\frac{\epsilon}{3.7D} - \frac{5.02}{Re}\log A_6\right]$$
$$A_6 = \frac{\epsilon}{3.7D} - \frac{5.02}{Re}\log A_5$$
$$A_5 = \frac{\epsilon}{3.7D} + \frac{13}{Re}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 4E-5 <= eD <= 5E-2

References

[R103], *[R104]*

Examples

```
>>> Zigrang_Sylvester_2(1E5, 1E-4)
0.01850021312358548
```

fluids.**Haaland**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Haaland (1983) *[R106]* as shown in *[R105]*.

$$f_f = \left(-1.8\log_{10}\left[\left(\frac{\epsilon/D}{3.7}\right)^{1.11} + \frac{6.9}{Re}\right]\right)^{-2}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 1E-6 <= eD <= 5E-2

References

[R105], *[R106]*

Examples

```
>>> Haaland(1E5, 1E-4)
0.018265053014793857
```

fluids.**Serghides_1**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Serghides (1984) *[R108]* as shown in *[R107]*.

$$f = \left[A - \frac{(B-A)^2}{C - 2B + A}\right]^{-2}$$
$$A = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{12}{Re}\right]$$
$$B = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51A}{Re}\right]$$
$$C = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51B}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R107], *[R108]*

Examples

```
>>> Serghides_1(1E5, 1E-4)
0.01851358983180063
```

fluids.**Serghides_2**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Serghides (1984) *[R110]* as shown in *[R109]*.

$$f_d = \left[4.781 - \frac{(A-4.781)^2}{B - 2A + 4.781}\right]^{-2}$$
$$A = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{12}{Re}\right]$$
$$B = -2\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51A}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R109], *[R110]*

Examples

```
>>> Serghides_2(1E5, 1E-4)
0.018486377560664482
```

fluids.**Tsal_1989**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Tsal (1989) *[R112]* as shown in *[R111]*.

$$A = 0.11(\frac{68}{Re} + \frac{\epsilon}{D})^{0.25}$$

if A >= 0.018 then fd = A if A < 0.018 then fd = 0.0028 + 0.85 A

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 0 <= eD <= 5E-2

References

[R111], *[R112]*

Examples

```
>>> Tsal_1989(1E5, 1E-4)
0.018382997825686878
```

fluids.**Manadilli_1997**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Manadilli (1997) *[R114]* as shown in *[R113]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left[\frac{\epsilon}{3.7D} + \frac{95}{Re^{0.983}} - \frac{96.82}{Re}\right]$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 5.245E3 <= Re <= 1E8; 0 <= eD <= 5E-2

References

[R113], *[R114]*

Examples

```
>>> Manadilli_1997(1E5, 1E-4)
0.01856964649724108
```

fluids.**Romeo_2002**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Romeo (2002) *[R116]* as shown in *[R115]*.

$$\frac{1}{\sqrt{f_d}} = -2\log\left\{\frac{\epsilon}{3.7065D} \times \frac{5.0272}{Re} \times \log\left[\frac{\epsilon}{3.827D} - \frac{4.567}{Re} \times \log\left(\frac{\epsilon}{7.7918D}^{0.9924} + \left(\frac{5.3326}{208.815 + Re}\right)^{0.9345}\right)\right]\right\}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 3E3 <= Re <= 1.5E8; 0 <= eD <= 5E-2

References

[R115], *[R116]*

Examples

```
>>> Romeo_2002(1E5, 1E-4)
0.018530291219676177
```

fluids.**Sonnad_Goudar_2006**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Sonnad and Goudar (2006) *[R118]* as shown in *[R117]*.

$$\frac{1}{\sqrt{f_d}} = 0.8686 \ln\left(\frac{0.4587Re}{S^{S/(S+1)}}\right)$$

$$S = 0.1240 \times \frac{\epsilon}{D} \times Re + \ln(0.4587Re)$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 4E3 <= Re <= 1E8; 1E-6 <= eD <= 5E-2

References

[R117], *[R118]*

Examples

```
>>> Sonnad_Goudar_2006(1E5, 1E-4)
0.0185971269898162
```

fluids.**Rao_Kumar_2007**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Rao and Kumar (2007) *[R120]* as shown in *[R119]*.

$$\frac{1}{\sqrt{f_d}} = 2\log\left(\frac{(2\frac{\epsilon}{D})^{-1}}{\left(\frac{0.444+0.135Re}{Re}\right)\beta}\right)$$

$$\beta = 1 - 0.55\exp(-0.33\ln\left[\frac{Re}{6.5}\right]^2)$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation. This equation is fit to original experimental friction factor data. Accordingly, this equation should not be used unless appropriate consideration is given.

References

[R119], *[R120]*

Examples

```
>>> Rao_Kumar_2007(1E5, 1E-4)
0.01197759334600925
```

fluids.**Buzzelli_2008**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Buzzelli (2008) *[R122]* as shown in *[R121]*.

$$\frac{1}{\sqrt{f_d}} = B_1 - \left[\frac{B_1 + 2\log(\frac{B_2}{Re})}{1 + \frac{2.18}{B_2}}\right]$$

$$B_1 = \frac{0.774\ln(Re) - 1.41}{1 + 1.32\sqrt{\frac{\epsilon}{D}}}$$

$$B_2 = \frac{\epsilon}{3.7D}Re + 2.51 \times B_1$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R121], *[R122]*

Examples

```
>>> Buzzelli_2008(1E5, 1E-4)
0.018513948401365277
```

fluids.**Avci_Karagoz_2009**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Avci and Karagoz (2009) *[R124]* as shown in *[R123]*.

$$f_D = \frac{6.4}{\left\{\ln(Re) - \ln\left[1 + 0.01Re\frac{\epsilon}{D}\left(1 + 10(\frac{\epsilon}{D})^{0.5}\right)\right]\right\}^{2.4}}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R123], *[R124]*

Examples

```
>>> Avci_Karagoz_2009(1E5, 1E-4)
0.01857058061066499
```

fluids.**Papaevangelo_2010**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Papaevangelo (2010) *[R126]* as shown in *[R125]*.

$$f_D = \frac{0.2479 - 0.0000947(7 - \log Re)^4}{\left[\log\left(\frac{\epsilon}{3.615D} + \frac{7.366}{Re^{0.9142}}\right)\right]^2}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 1E4 <= Re <= 1E7; 1E-5 <= eD <= 1E-3

References

[R125], *[R126]*

Examples

```
>>> Papaevangelo_2010(1E5, 1E-4)
0.015685600818488177
```

fluids.**Brkic_2011_1**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Brkic (2011) *[R128]* as shown in *[R127]*.

$$f_d = [-2\log(10^{-0.4343\beta} + \frac{\epsilon}{3.71D})]^{-2}$$

$$\beta = \ln\frac{Re}{1.816\ln\left(\frac{1.1Re}{\ln(1+1.1Re)}\right)}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R127], *[R128]*

Examples

```
>>> Brkic_2011_1(1E5, 1E-4)
0.01812455874141297
```

fluids.**Brkic_2011_2**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Brkic (2011) *[R130]* as shown in *[R129]*.

$$f_d = [-2\log(\frac{2.18\beta}{Re} + \frac{\epsilon}{3.71D})]^{-2}$$

$$\beta = \ln\frac{Re}{1.816\ln\left(\frac{1.1Re}{\ln(1+1.1Re)}\right)}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

No range of validity specified for this equation.

References

[R129], *[R130]*

Examples

```
>>> Brkic_2011_2(1E5, 1E-4)
0.018619745410688716
```

fluids.**Fang_2011**(*Re*, *eD*)

Calculates Darcy friction factor using the method in Fang (2011) *[R132]* as shown in *[R131]*.

$$f_D = 1.613\left\{\ln\left[0.234\frac{\epsilon}{D}^{1.1007} - \frac{60.525}{Re^{1.1105}} + \frac{56.291}{Re^{1.0712}}\right]\right\}^{-2}$$

Parameters **Re** : float

Reynolds number, [-]

eD : float

Relative roughness, [-]

Returns **fd** : float

Darcy friction factor [-]

Notes

Range is 3E3 <= Re <= 1E8; 0 <= eD <= 5E-2

References

[R131], *[R132]*

Examples

```
>>> Fang_2011(1E5, 1E-4)
0.018481390682985432
```

class fluids.**TANK**(*D=None*, *L=None*, *horizontal=True*, *sideA=None*, *sideB=None*, *sideA_a=0*, *sideB_a=0*, *sideA_f=1.0*, *sideA_k=0.06*, *sideB_f=1.0*, *sideB_k=0.06*, *sideA_a_ratio=0.25*, *sideB_a_ratio=0.25*, *L_over_D=None*, *V=None*)

Bases: object

Class representing tank volumes and levels. All parameters are also attributes.

Examples

Total volume of a tank:

```
>>> TANK(D=1.2, L=4, horizontal=False).V_total
4.523893421169302
```

Volume of a tank at a given height:

```
>>> TANK(D=1.2, L=4, horizontal=False).V_from_h(.5)
0.5654866776461628
```

Height of liquid for a given volume:

```
>>> TANK(D=1.2, L=4, horizontal=False).h_from_V(.5)
0.44209706414415373
```

Solving for tank volumes, first horizontal, then vertical:

```
>>> TANK(D=10., horizontal=True, sideA='conical', sideB='conical', V=500).L
4.699531057009146
>>> TANK(L=4.69953105701, horizontal=True, sideA='conical', sideB='conical', V=500).D
9.999999999999407
>>> TANK(L_over_D=0.469953105701, horizontal=True, sideA='conical', sideB='conical', V=500).L
4.69953105700979
```

```
>>> TANK(D=10., horizontal=False, sideA='conical', sideB='conical', V=500).L
4.699531057009147
>>> TANK(L=4.69953105701, horizontal=False, sideA='conical', sideB='conical', V=500).D
9.999999999999407
>>> TANK(L_over_D=0.469953105701, horizontal=False, sideA='conical', sideB='conical', V=500).L
4.69953105700979
```

Attributes

table	(bool) Whether or not a table of heights-volumes has been generated
h_max	(float) Height of the tank, [m]
V_total	(float) Total volume of the tank as calculated [m^3]
heights	(ndarray) Array of heights between 0 and h_max, [m]
volumes	(ndarray) Array of volumes calculated from the heights [m^3]

Methods

V_from_h(*h*)

Method to calculate the volume of liquid in a fully defined tank given a specified height *h*. *h* must be under the maximum height.

Parameters **h** : float

Height specified, [m]

Returns **V** : float

Volume of liquid in the tank up to the specified height, [m^3]

h_from_V(*V*)

Method to calculate the height of liquid in a fully defined tank given a specified volume of liquid in it *V*.

V must be under the maximum volume. If interpolation table is not yet defined, creates it by calling the method set_table.

Parameters **V** : float

Volume of liquid in the tank up to the desired height, [m^3]

Returns **h** : float

Height of liquid at which the volume is as desired, [m]

set_misc()

Set more parameters, after the tank is better defined than in the __init__ function.

Notes

Two of D, L, and L_over_D must be known when this function runs. The other one is set from the other two first thing in this function. a_ratio parameters are used to calculate a values for the heads here, if applicable. Radius is calculated here. Maximum tank height is calculated here. V_total is calculated here.

set_table(*n=100*, *dx=None*)

Method to set an interpolation table of liquids levels versus volumes in the tank, for a fully defined tank. Normally run by the h_from_V method, this may be run prior to its use with a custom specification. Either the number of points on the table, or the vertical distance between steps may be specified.

Parameters **n** : float, optional

Number of points in the interpolation table, [-]

dx : float, optional

Vertical distance between steps in the interpolation table, [m]

solve_tank_for_V()

Method which is called to solve for tank geometry when a certain volume is specified. Will be called by the __init__ method if V is set.

Notes

Raises an error if L and either of sideA_a or sideB_a are specified; these can only be set once D is known. Raises an error if more than one of D, L, or L_over_D are specified. Raises an error if the head ratios are not provided.

Calculates initial guesses assuming no heads are present, and then uses fsolve to determine the correct dimentions for the tank.

Tested, but bugs and limitations are expected here.

table = False

fluids.**adjust_homogeneity**(*fraction*)

Base: 95% homogeneity

fluids.**agitator_time_homogeneous**(*D=None*, *N=None*, *P=None*, *T=None*, *H=None*, *mu=None*, *rho=None*, *homogeneity=0.95*)

Calculates time for a fluid mizing in a tank with an impeller to reach a specified level of homogeneity, according

to *[R133]*.

$$N_p = \frac{Pg}{\rho N^3 D^5}$$

$$Re_{imp} = \frac{\rho D^2 N}{\mu}$$

$$\text{constant} = N_p^{1/3} Re_{imp}$$

$$Fo = 5.2/\text{constant} \text{for turbulent regime}$$

$$Fo = (183/\text{constant})^2 \text{for transition regime}$$

Parameters **D** : float

Impeller diameter (optional) [m]

N : float:

Speed of impeller, [r/s]

P : float

Actual power required to mix, ignoring mechanical inefficiencies [W]

T : float

Tank diameter, [m]

H : float

Tank height, [m]

mu : float

Mixture viscosity, [Pa*s]

rho : float

Mixture density, [kg/m^3]

homogeneity : float

Fraction completion of mixing, optional, []

Returns **t** : float

Time for specified degree of homogeneity [s]

Notes

If impeller diameter is not specified, assumed to be 0.5 tank diameters.

The first example is solved forward rather than backwards here. A rather different result is obtained, but is accurate.

No check to see if the mixture if laminar is currently implemented. This would underpredict the required time.

References

[R133]

Examples

```
>>> agitator_time_homogeneous(D=36*.0254, N=56/60., P=957., T=1.83, H=1.83, mu=0.018, rho=1020,
15.143198226374668
```

```
>>> agitator_time_homogeneous(D=1, N=125/60., P=298., T=3, H=2.5, mu=.5, rho=980, homogeneity=.9
67.7575069865228
```

fluids.**Kp_helical_ribbon_Rieger**(*D=None*, *h=None*, *nb=None*, *pitch=None*, *width=None*, *T=None*)

Calculates product of power number and reynolds number for a specified geometry for a heilical ribbon mixer in the laminar regime. One of several correlations listed in *[R134]*, it used more data than other listed correlations and was recommended.

$$K_p = 82.8\frac{h}{D}\left(\frac{c}{D}\right)^{-0.38}\left(\frac{p}{D}\right)^{-0.35}\left(\frac{w}{D}\right)^{0.20} n_b^{0.78}$$

Parameters **D** : float

Impeller diameter (optional) [m]

h : float

Ribbon mixer height, [m]

nb : float:

Number of blades, [-]

pitch : float

Height of one turn around a helix [m]

width : float

Width of one blade [m]

T : float

Tank diameter, [m]

Returns **Kp** : float

Product of power number and reynolds number for laminar regime []

Notes

Example is from example 9-6 in *[R134]*. Confirmed.

References

[R134], *[R135]*

Examples

```
>>> Kp_helical_ribbon_Rieger(D=1.9, h=1.9, nb=2, pitch=1.9, width=.19, T=2)
357.39749163259256
```

fluids.**time_helical_ribbon_Grenville**(*Kp*, *N*)

Calculates product of time required for mixing in a helical ribbon coil in the laminar regime according to the Grenville *[R137]* method recommended in *[R136]*.

$$t = 896 \times 10^3 K_p^{-1.69}/N$$

Parameters **Kp** : float

Product of power number and reynolds number for laminar regime []

N : float:

Speed of impeller, [r/s]

Returns **t** : float

Time for homogeneity [s]

Notes

Degree of homogeneity is not specified. Example is from example 9-6 in *[R136]*. Confirmed.

References

[R136], *[R137]*

Examples

```
>>> time_helical_ribbon_Grenville(357.4, 4/60.)
650.980654028894
```

fluids.**size_tee**(*Q1=None*, *Q2=None*, *D=None*, *D2=None*, *n=1*, *pipe_diameters=5*)

Calculates CoV of an optimal or specified tee for mixing at a tee according to *[R138]*. Assumes turbulent flow. The smaller stream in injected into the main pipe, which continues straight. COV calculation is according to *[R139]*.

$$TODO$$

Parameters **Q1** : float

Volumetric flow rate of larger stream [m^3/s]

Q2 : float

Volumetric flow rate of smaller stream [m^3/s]

D : float

Diameter of pipe after tee [m]

D2 : float

Diameter of mixing inlet, optional (optimally calculated if not specified) [m]

n : float

Number of jets, 1 to 4 []

pipe_diameters : float

Number of diameters along tail pipe for CoV calculation, 0 to 5 []

Returns **CoV** : float

Standard deviation of dimentionless concentration [-]

Notes

Not specified if this works for liquid also, though probably not. Example is from example Example 9-6 in *[R138]*. Low precision used in example.

References

[R138], *[R139]*

Examples

```
>>> size_tee(Q1=11.7, Q2=2.74, D=0.762, D2=None, n=1, pipe_diameters=5)
0.2940930233038544
```

fluids.**COV_motionless_mixer**(*Ki=None*, *Q1=None*, *Q2=None*, *pipe_diameters=None*)

Calculates CoV of a motionless mixer with a regression parameter in *[R140]* and originally in *[R141]*.

$$\frac{CoV}{CoV_0} = K_i^{L/D}$$

Parameters **Ki** : float

Correlation parameter specific to a mixer's design, [-]

Q1 : float

Volumetric flow rate of larger stream [m^3/s]

Q2 : float

Volumetric flow rate of smaller stream [m^3/s]

pipe_diameters : float

Number of diameters along tail pipe for CoV calculation, 0 to 5 []

Returns **CoV** : float

Standard deviation of dimentionless concentration [-]

Notes

Example 7-8.3.2 in *[R140]*, solved backwards.

References

[R140], *[R141]*

Examples

```
>>> COV_motionless_mixer(Ki=.33, Q1=11.7, Q2=2.74, pipe_diameters=4.74/.762)
0.0020900028665727685
```

fluids.**K_motionless_mixer**(*K=None*, *L=None*, *D=None*, *fd=None*)

Calculates loss ciefficient of a motionless mixer with a regression parameter in *[R142]* and originally in *[R143]*.

$$K = K_{L/T} f \frac{L}{D}$$

Parameters **K** : float

Correlation parameter specific to a mixer's design, [-] Also specific to laminar or turbulent regime.

L : float

Length of the motionless mixer [m]

D : float

Diameter of pipe [m]

fd : float

Darcy friction factor [-]

Returns **K** : float

Loss coefficient of mixer [-]

Notes

Related to example 7-8.3.2 in *[R142]*.

References

[R142], *[R143]*

Examples

```
>>> K_motionless_mixer(K=150, L=.762*5, D=.762, fd=.01)
7.5
```

fluids.**Q_weir_V_Shen**(*h1*, *angle=90*)

Calculates the flow rate across a V-notch (triangular) weir from the height of the liquid above the tip of the notch, and with the angle of the notch. Most of these type of weir are 90 degrees. Model from *[R144]* as reproduced in *[R145]*.

Flow rate is given by:

$$Q = C \tan\left(\frac{\theta}{2}\right) \sqrt{g}(h_1 + k)^{2.5}$$

Parameters **h1** : float

Height of the fluid above the notch [m]

angle : float, optional

Angle of the notch [degrees]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

angles = [20, 40, 60, 80, 100] Cs = [0.59, 0.58, 0.575, 0.575, 0.58] k = [0.0028, 0.0017, 0.0012, 0.001, 0.001]

The following limits apply to the use of this equation:

h1 >= 0.05 m h2 > 0.45 m h1/h2 <= 0.4 m b > 0.9 m

$$\frac{h_1}{b}\tan\left(\frac{\theta}{2}\right) < 2$$

Flows are lower than obtained by the curves at http://www.lmnoeng.com/Weirs/vweir.php.

References

[R144], *[R145]*

Examples

```
>>> Q_weir_V_Shen(0.6, angle=45)
0.21071725775478228
>>> Q_weir_V_Shen(1.2)
2.8587083148501078
```

fluids.**Q_weir_rectangular_Smith**(*h1*, *b*, *h2=None*, *b1=None*, *V1=None*)

fluids.**Q_weir_rectangular_Kindsvater_Carter**(*h1*, *h2*, *b*)

Calculates the flow rate across rectangular weir from the height of the liquid above the crest of the notch, the liquid depth beneath it, and the width of the notch. Model from *[R146]* as reproduced in *[R147]*.

Flow rate is given by:

$$Q = 0.554\left(1 - 0.0035\frac{h_1}{h_2}\right)(b + 0.0025)\sqrt{g}(h_1 + 0.0001)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the rectangular flow section of the weir [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

b/b1 0.2 h1/h2 < 2 b > 0.15 m h1 > 0.03 m h2 > 0.1 m

References

[R146], *[R147]*

Examples

```
>>> Q_weir_rectangular_Kindsvater_Carter(0.2, 0.5, 1)
0.15545928949179422
```

fluids.**Q_weir_rectangular_SIA**(*h1*, *h2*, *b*, *b1*)

Calculates the flow rate across rectangular weir from the height of the liquid above the crest of the notch, the liquid depth beneath it, and the width of the notch. Model from *[R148]* as reproduced in *[R149]*.

Flow rate is given by:

$$Q = 0.544\left[1 + 0.064\left(\frac{b}{b_1}\right)^2 + \frac{0.00626 - 0.00519(b/b_1)^2}{h_1 + 0.0016}\right]\left[1 + 0.5\left(\frac{b}{b_1}\right)^4\left(\frac{h_1}{h_1 + h_2}\right)^2\right] b\sqrt{g}h^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the rectangular flow section of the weir [m]

b1 : float

Width of the full section of the channel [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

b/b1 0.2 h1/h2 < 2 b > 0.15 m h1 > 0.03 m h2 > 0.1 m

References

[R148], *[R149]*

Examples

```
>>> Q_weir_rectangular_SIA(0.2, 0.5, 1, 2)
1.0408858453811165
```

fluids.**Q_weir_rectangular_full_Ackers**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R150]* as reproduced in *[R151]*, confirmed with *[R152]*.

Flow rate is given by:

$$Q = 0.564\left(1 + 0.150\frac{h_1}{h_2}\right) b\sqrt{g}(h_1 + 0.001)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

h1 > 0.02 m h2 > 0.15 m h1/h2 2.2

References

[R150], *[R151]*, *[R152]*

Examples

Example as in *[R152]*, matches. However, example is unlikely in practice.

```
>>> Q_weir_rectangular_full_Ackers(h1=0.9, h2=0.6, b=5)
9.251938159899948
```

```
>>> Q_weir_rectangular_full_Ackers(h1=0.3, h2=0.4, b=2)
0.6489618999846898
```

fluids.**Q_weir_rectangular_full_SIA**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R153]* as reproduced in *[R154]*.

Flow rate is given by:

$$Q = \frac{2}{3}\sqrt{2}\left(0.615 + \frac{0.000615}{h_1 + 0.0016}\right) b\sqrt{g}h_1 + 0.5\left(\frac{h_1}{h_1 + h_2}\right)^2 b\sqrt{g}h_1^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

0.025 < h < 0.8 m b > 0.3 m h2 > 0.3 m h1/h2 < 1

References

[R153], *[R154]*

Examples

Example compares terribly with the Ackers expression - probable error in *[R154]*. DO NOT USE.

```
>>> Q_weir_rectangular_full_SIA(h1=0.3, h2=0.4, b=2)
1.1875825055400384
```

fluids.**Q_weir_rectangular_full_Rehbock**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R156]* as reproduced in *[R157]*.

Flow rate is given by:

$$Q = \frac{2}{3}\sqrt{2}\left(0.602 + 0.0832\frac{h_1}{h_2}\right) b\sqrt{g}(h_1 + 0.00125)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

0.03 m < h1 < 0.75 m b > 0.3 m h2 > 0.3 m h1/h2 < 1

References

[R156], *[R157]*

Examples

```
>>> Q_weir_rectangular_full_Rehbock(h1=0.3, h2=0.4, b=2)
0.6486856330601333
```

fluids.**Q_weir_rectangular_full_Kindsvater_Carter**(*h1*, *h2*, *b*)

Calculates the flow rate across a full-channel rectangular weir from the height of the liquid above the crest of the weir, the liquid depth beneath it, and the width of the channel. Model from *[R158]* as reproduced in *[R159]*.

Flow rate is given by:

$$Q = \frac{2}{3}\sqrt{2}\left(0.602 + 0.0832\frac{h_1}{h_2}\right) b\sqrt{g}(h_1 + 0.00125)^{1.5}$$

Parameters **h1** : float

Height of the fluid above the crest of the weir [m]

h2 : float

Height of the fluid below the crest of the weir [m]

b : float

Width of the channel section [m]

Returns **Q** : float

Volumetric flow rate across the weir [m^3/s]

Notes

The following limits apply to the use of this equation:

h1 > 0.03 m b > 0.15 m h2 > 0.1 m h1/h2 < 2

References

[R158], *[R159]*

Examples

```
>>> Q_weir_rectangular_full_Kindsvater_Carter(h1=0.3, h2=0.4, b=2)
0.641560300081563
```

fluids.**V_Manning**(*Rh*, *S*, *n*)

Predicts the average velocity of a flow across an open channel of hydraulic radius Rh and slope S, given the Manning roughness coefficient n.

Flow rate is given by:

$$V = \frac{1}{n} R_h^{2/3} S^{0.5}$$

Parameters **Rh** : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

S : float

Slope of the channel, m/m [-]

n : float

Manning roughness coefficient [s/m^(1/3)]

Returns **V** : float

Average velocity of the channel [m/s]

Notes

This is equation is often given in imperial units multiplied by 1.49.

References

[R160], *[R161]*

Examples

Example is from *[R161]*, matches.

```
>>> V_Manning(0.2859, 0.005236, 0.03)*0.5721
0.5988618058239864
```

Custom example, checked.

```
>>> V_Manning(Rh=5, S=0.001, n=0.05)
1.8493111942973235
```

fluids.**n_Manning_to_C_Chezy**(*n*, *Rh*)

Converts a Manning roughness coefficient to a Chezy coefficient, given the hydraulic radius of the channel.

$$C = \frac{1}{n} R_h^{1/6}$$

Parameters **n** : float

Manning roughness coefficient [s/m^(1/3)]

Rh : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

Returns **C** : float

Chezy coefficient [m^0.5/s]

References

[R162]

Examples

Custom example, checked.

```
>>> n_Manning_to_C_Chezy(0.05, Rh=5)
26.15320972023661
```

fluids.**C_Chezy_to_n_Manning**(*C*, *Rh*)

Converts a Chezy coefficient to a Manning roughness coefficient, given the hydraulic radius of the channel.

$$n = \frac{1}{C} R_h^{1/6}$$

Parameters **C** : float

Chezy coefficient [m^0.5/s]

Rh : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

Returns **n** : float

Manning roughness coefficient [s/m^(1/3)]

References

[R163]

Examples

Custom example, checked.

```
>>> C_Chezy_to_n_Manning(26.15, Rh=5)
0.05000613713238358
```

fluids.**V_Chezy**(*Rh*, *S*, *C*)

Predicts the average velocity of a flow across an open channel of hydraulic radius Rh and slope S, given the Chezy coefficient C.

Flow rate is given by:

$$V = C\sqrt{SR_h}$$

Parameters **Rh** : float

Hydraulic radius of the channel, Flow Area/Wetted perimeter [m]

S : float

Slope of the channel, m/m [-]

C : float

Chezy coefficient [m^0.5/s]

Returns **V** : float

Average velocity of the channel [m/s]

References

[R164], *[R165]*, *[R166]*

Examples

Custom example, checked.

```
>>> V_Chezy(Rh=5, S=0.001, C=26.153)
1.8492963648371776
```

fluids.**Ergun**(*Dp*, *voidage=0.4*, *sphericity=1*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed, using the famous Ergun equation.

Pressure drop is given by:

$$\Delta p = \frac{150\mu(1-\epsilon)^2 V_s L}{\epsilon^3(\Phi D_p)^2} + \frac{1.75(1-\epsilon)\rho V_s^2 L}{\epsilon^3(\Phi D_p)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

sphericity : float

Sphericity of particles in bed []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

The first term in this equation represents laminar loses, and the second, turbulent loses. Sphericity must be calculated. According to *[R167]*, developed using spheres, pulverized coke/coal, sand, cylinders and tablets for ranges of $1 < RE_{ERg} < 2300$. *[R167]* cites a source claiming it should not be used above 500.

References

[R167]

Examples

```
>>> # Custom example
>>> Ergun(Dp=0.0008, voidage=0.4, sphericity=1., H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-00
892.3013355913797
```

fluids.**Kuo_Nydegger**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R169]*, originally in *[R168]*.

Pressure drop is given by:

$$\frac{\Delta P}{L}\frac{D_p^2}{\mu v_s}\left(\frac{\phi}{1-\phi}\right)^2 = 276 + 5.05Re^{0.87}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

Does not share exact form with the Ergun equation. 767 < Re_{ergun} < 24330

References

[R168], *[R169]*

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Kuo_Nydegger(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
1658.3187666274648
```

Re = 4000 custom example:

```
>>> Kuo_Nydegger(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
241.56063171630015
```

fluids.**Jones_Krier**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R170]*.

Pressure drop is given by:

$$\frac{\Delta P}{L}\frac{D_p^2}{\mu v_s}\left(\frac{\phi}{1-\phi}\right)^2 = 150 + 1.89Re^{0.87}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

Does not share exact form with the Ergun equation. 733 < Re_{ergun} < 126670

References

[R170]

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Jones_Krier(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
911.494265366317
```

Re = 4000 custom example:

```
>>> Jones_Krier(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
184.69401245425462
```

fluids.**Carman**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R172]*, originally in *[R171]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = \frac{180}{Re_{Erg}} + \frac{2.87}{Re_{Erg}^{0.1}}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$0.1 < RE_{ERg} < 60000$

References

[R171], *[R172]*

Examples

Custom example:

```
>>> Carman(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
1077.0587868633704
```

Re = 4000 custom example:

```
>>> Carman(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
178.2490332160841
```

fluids.**Hicks**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R174]*, originally in *[R173]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = \frac{6.8}{Re_{Erg}^{0.2}}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$300 < RE_{ERg} < 60000$

References

[R173], *[R174]*

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Hicks(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
62.534899706155834
```

Re = 4000 custom example:

```
>>> Hicks(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
171.20579747453397
```

fluids.**Brauer**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R176]*, originally in *[R175]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = \frac{160}{Re_{Erg}} + \frac{3.1}{Re_{Erg}^{0.1}}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$0.01 < RE_{ERg} < 40000$

References

[R175], *[R176]*

Examples

Custom example:

```
>>> Brauer(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
962.7294566294247
```

Re = 4000 custom example:

```
>>> Brauer(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
191.77738833880164
```

fluids.**Montillet**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*, *Dc=None*)

Calculates pressure drop across a packed bed of spheres as in *[R178]*, originally in *[R177]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p \frac{\epsilon^3}{(1-\epsilon)} = a\left(\frac{D_c}{D_p}\right)^{0.20}\left(\frac{1000}{Re_p} + \frac{60}{Re_p^{0.5}} + 12\right)$$

$$Re_p = \frac{\rho v_s D_p}{\mu}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Dc : float, optional

Diameter of the column, [m]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$10 < REp < 2500$ if Dc/D > 50, set to 2.2. a = 0.061 for epsilon < 0.4, 0.050 for > 0.4.

References

[R177], *[R178]*

Examples

Custom example:

```
>>> Montillet(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
1148.1905244077548
```

Re = 4000 custom example:

```
>>> Montillet(Dp=0.08, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
212.67409611116554
```

fluids.**Idelchik**(*Dp*, *voidage=0.4*, *H=None*, *vs=None*, *rho=None*, *mu=None*)

Calculates pressure drop across a packed bed of spheres as in *[R180]*, originally in *[R179]*.

Pressure drop is given by:

$$\frac{\Delta P}{L\rho V_s^2} D_p = \frac{0.765}{\epsilon^{4.2}}\left(\frac{30}{Re_l} + \frac{3}{Re_l^{0.7}} + 0.3\right)$$

$$Re_l = (0.45/\epsilon^{0.5}) Re_{Erg}$$

$$Re_{Erg} = \frac{\rho v_s D_p}{\mu(1-\epsilon)}$$

Parameters **Dp** : float

Particle diameter [m]

voidage : float

Void fraction of bed packing []

H : float

Height of packed bed [m]

vs : float

Superficial velocity of fluid [m/s]

rho : float

Density of fluid [kg/m^3]

mu : float

Viscosity of fluid, [Pa*S]

Returns **dP** : float

Pressure drop across bed [Pa]

Notes

$0.001 < RE_{ERg} < 1000$ This equation is valid for void fractions between 0.3 and 0.8. Cited as by Bernshtein. This model is likely presented in *[R180]* with a typo, as it varries greatly from other models.

References

[R179], *[R180]*

Examples

Custom example, outside lower limit of Re (Re = 1):

```
>>> Idelchik(Dp=0.0008, voidage=0.4, H=0.5, vs=0.00132629120, rho=1000., mu=1.00E-003)
56.13431404382615
```

Re = 400 custom example:

```
>>> Idelchik(Dp=0.008, voidage=0.4, H=0.5, vs=0.05, rho=1000., mu=1.00E-003)
120.55068459098145
```

fluids.**voidage_Benyahia_Oneil**(*Dpe*, *Dt*, *sphericity*)

Calculates voidage of a bed of arbitraryily shaped uniform particles packed into a bed or tube of diameter *Dt*, with equivalent sphere diameter *Dp*. Shown in *[R181]*, and cited by various authors. Correlations exist also for spheres, solid cylinders, hollow cylinders, and 4-hole cylinders. Based on a series of physical measurements.

$$\epsilon = 0.1504 + \frac{0.2024}{\phi} + \frac{1.0814}{\left(\frac{d_t}{d_{pe}} + 0.1226\right)^2}$$

Parameters **Dpe** : float

Equivalent spherical particle diameter, [m]

Dt : float

Diameter of the tube, [m]

sphericity : float

Sphericity of particles in bed []

Returns **voidage** : float

Void fraction of bed packing []

Notes

Average error of 5.2%; valid 1.5 < dtube/dp < 50 and 0.42 < sphericity < 1

References

[R181]

Examples

```
>>> voidage_Benyahia_Oneil(1E-3, 1E-2, .8)
0.41395363849210065
```

fluids.**voidage_Benyahia_Oneil_spherical**(*Dp*, *Dt*)

Calculates voidage of a bed of spheres packed into a bed or tube of diameter *Dt*, with sphere diameters *Dp*. Shown in *[R182]*, and cited by various authors. Correlations exist also for solid cylinders, hollow cylinders, and 4-hole cylinders. Based on a series of physical measurements.

$$\epsilon = 0.390 + \frac{1.740}{\left(\frac{d_{cyl}}{d_p} + 1.140\right)^2}$$

Parameters **Dp** : float

Spherical particle diameter, [m]

Dt : float

Diameter of the tube, [m]

Returns **voidage** : float

Void fraction of bed packing []

Notes

Average error 1.5%, 1.5 < ratio < 50.

References

[R182]

Examples

```
>>> voidage_Benyahia_Oneil_spherical(.001, .05)
0.3906653157443224
```

fluids.**voidage_Benyahia_Oneil_cylindrical**(*Dpe*, *Dt*, *sphericity*)

Calculates voidage of a bed of cylindrical uniform particles packed into a bed or tube of diameter *Dt*, with equivalent sphere diameter *Dpe*. Shown in *[R183]*, and cited by various authors. Correlations exist also for spheres, solid cylinders, hollow cylinders, and 4-hole cylinders. Based on a series of physical measurements.

$$\epsilon = 0.373 + \frac{1.703}{\left(\frac{d_{cyl}}{d_p} + 0.611\right)^2}$$

Parameters **Dpe** : float

Equivalent spherical particle diameter, [m]

Dt : float

Diameter of the tube, [m]

sphericity : float

Sphericity of particles in bed []

Returns **voidage** : float

Void fraction of bed packing []

Notes

Average error 0.016%; 1.7 < ratio < 26.3.

References

[R183]

Examples

```
>>> voidage_Benyahia_Oneil_cylindrical(.01, .1, .6)
0.38812523109607894
```

fluids.**nearest_pipe**(*Do=None*, *Di=None*, *NPS=None*, *schedule='40'*)

Searches for and finds the nearest standard pipe size to a given specification. Acceptable inputs are:

- Nominal pipe size
- Nominal pipe size and schedule
- Outer diameter *Do*
- Outer diameter *Do* and schedule
- Inner diameter *Di*
- Inner diameter *Di* and schedule

Acceptable schedules are: '5', '10', '20', '30', '40', '60', '80', '100', '120', '140', '160', 'STD', 'XS', 'XXS', '5S', '10S', '40S', '80S'.

Parameters **Do** : float

Pipe outer diameter, [m]

Di : float

Pipe inner diameter, [m]

NPS : float

Nominal pipe size, []

schedule : str

String representing schedule size

Returns **NPS** : float

Nominal pipe size, []

_di : float

Pipe inner diameter, [m]

_do : float

Pipe outer diameter, [m]

_t : float

Pipe wall thickness, [m]

Notes

Internal units within this function are mm. The imperial schedules are not quite identical to these value, but all rounding differences happen in the sub-0.1 mm level.

References

[R184], *[R185]*

Examples

```
>>> nearest_pipe(Di=0.021)
(1, 0.02664, 0.0334, 0.0033799999999999998)
>>> nearest_pipe(Do=.273, schedule='5S')
(10, 0.26630000000000004, 0.2731, 0.0034)
```

fluids.**gauge_from_t**(*t*, *SI=True*, *schedule='BWG'*)

Looks up the gauge of a given wire thickness of given schedule. Values are all non-linear, and tabulated internally.

Parameters **t** : float

Thickness, [m]

SI : bool, optional

If False, value in inches is returned, rather than m.

schedule : str

Gauge schedule, one of 'BWG', 'AWG', 'SWG', 'MWG', 'BSWG', or 'SSWG'

Returns **gauge** : float-like

Wire Gauge, []

Notes

Birmingham Wire Gauge (BWG) ranges from 0.2 (0.5 inch) to 36 (0.004 inch).

American Wire Gauge (AWG) ranges from 0.167 (0.58 inch) to 51 (0.00099 inch). These are used for electrical wires.

Steel Wire Gauge (SWG) ranges from 0.143 (0.49 inch) to 51 (0.0044 inch). Also called Washburn & Moen wire gauge, American Steel gauge, Wire Co. gauge, and Roebling wire gauge.

Music Wire Gauge (MWG) ranges from 0.167 (0.004 inch) to 46 (0.18 inch). Also called Piano Wire Gauge.

British Standard Wire Gage (BSWG) ranges from 0.143 (0.5 inch) to 51 (0.001 inch). Also called Imperial Wire Gage (IWG).

Stub's Steel Wire Gage (SSWG) ranges from 1 (0.227 inch) to 80 (0.013 inch)

References

[R186]

Examples

```
>>> gauge_from_t(.5, False, 'BWG'), gauge_from_t(0.005588, True)
(0.2, 5)
>>> gauge_from_t(0.5165, False, 'AWG'), gauge_from_t(0.00462026, True, 'AWG')
(0.2, 5)
>>> gauge_from_t(.4305, False, 'SWG'), gauge_from_t(0.0052578, True, 'SWG')
(0.2, 5)
>>> gauge_from_t(.005, False, 'MWG'), gauge_from_t(0.0003556, True, 'MWG')
(0.2, 5)
```

```
    >>> gauge_from_t(.432, False, 'BSWG'), gauge_from_t(0.0053848, True, 'BSWG')
    (0.2, 5)
    >>> gauge_from_t(0.227, False, 'SSWG'), gauge_from_t(0.0051816, True, 'SSWG')
    (1, 5)
```

fluids.**t_from_gauge**(*gauge*, *SI=True*, *schedule='BWG'*)

Looks up the thickness of a given wire gauge of given schedule. Values are all non-linear, and tabulated internally.

Parameters **gauge** : float-like

Wire Gauge, []

SI : bool, optional

If False, value in inches is returned, rather than m.

schedule : str

Gauge schedule, one of 'BWG', 'AWG', 'SWG', 'MWG', 'BSWG', or 'SSWG'

Returns **t** : float

Thickness, [m]

Notes

Birmingham Wire Gauge (BWG) ranges from 0.2 (0.5 inch) to 36 (0.004 inch).

American Wire Gauge (AWG) ranges from 0.167 (0.58 inch) to 51 (0.00099 inch). These are used for electrical wires.

Steel Wire Gauge (SWG) ranges from 0.143 (0.49 inch) to 51 (0.0044 inch). Also called Washburn & Moen wire gauge, American Steel gauge, Wire Co. gauge, and Roebling wire gauge.

Music Wire Gauge (MWG) ranges from 0.167 (0.004 inch) to 46 (0.18 inch). Also called Piano Wire Gauge.

British Standard Wire Gage (BSWG) ranges from 0.143 (0.5 inch) to 51 (0.001 inch). Also called Imperial Wire Gage (IWG).

Stub's Steel Wire Gage (SSWG) ranges from 1 (0.227 inch) to 80 (0.013 inch)

References

[R187]

Examples

```
    >>> t_from_gauge(.2, False, 'BWG'), t_from_gauge(5, True)
    (0.5, 0.005588)
    >>> t_from_gauge(.2, False, 'AWG'), t_from_gauge(5, True, 'AWG')
    (0.5165, 0.00462026)
    >>> t_from_gauge(.2, False, 'SWG'), t_from_gauge(5, True, 'SWG')
    (0.4305, 0.0052578)
    >>> t_from_gauge(.2, False, 'MWG'), t_from_gauge(5, True, 'MWG')
    (0.005, 0.0003556)
    >>> t_from_gauge(.2, False, 'BSWG'), t_from_gauge(5, True, 'BSWG')
    (0.432, 0.0053848)
```

```
>>> t_from_gauge(1, False, 'SSWG'), t_from_gauge(5, True, 'SSWG')
(0.227, 0.0051816)
```

fluids.**VFD_efficiency**(*P*, *load=1*)

Returns the efficiency of a Variable Frequency Drive according to *[R188]*. These values are generic, and not standardized as minimum values. Older VFDs often have much worse performance.

Parameters **P** : float

Power, [W]

load : float, optional

Fraction of motor's rated electrical capacity being used

Returns **effciency** : float

VFD efficiency, [-]

Notes

The use of a VFD does change the characteristics of a pump curve's efficiency, but this has yet to be quantified. The effect is small. This value should be multiplied by the product of the pump and motor efficiency to determine the overall efficiency.

Efficiency table is in units of hp, so a conversion is performed internally. If load not specified, assumed 1 - where maximum efficiency occurs. Table extends down to 3 hp and up to 400 hp; values outside these limits are rounded to the nearest known value. Values between standardized sizes are interpolated linearly. Load values extend down to 0.016.

References

[R188]

Examples

```
>>> VFD_efficiency(10*hp)
0.96
>>> VFD_efficiency(100*hp, load=0.5)
0.96
```

fluids.**CSA_motor_efficiency**(*P*, *closed=False*, *poles=2*, *high_efficiency=False*)

Returns the efficiency of a NEMA motor according to *[R189]*. These values are standards, but are only for full-load operation.

Parameters **P** : float

Power, [W]

closed : bool, optional

Whether or not the motor is enclosed

poles : int, optional

The number of poles of the motor

high_efficiency : bool, optional

Whether or not to look up the high-efficiency value

Returns **effciency** : float

Guaranteed full-load motor efficiency, [-]

Notes

Criteria for being required to meet the high-efficiency standard is:

- Designed for continuous operation
- Operates by three-phase induction
- Is a squirrel-cage or cage design
- Is NEMA type A, B, or C with T or U frame; or IEC design N or H
- Is designed for single-speed operation
- Has a nominal voltage of less than 600 V AC
- Has a nominal frequency of 60 Hz or 50/60 Hz
- Has 2, 4, or 6 pole construction
- Is either open or closed

Pretty much every motor is required to meet the low-standard efficiency table, however.

Several low-efficiency standard high power values were added to allow for easy programming; values are the last listed efficiency in the table.

References

[R189]

Examples

```
>>> CSA_motor_efficiency(100*hp)
0.93
>>> CSA_motor_efficiency(100*hp, closed=True, poles=6, high_efficiency=True)
0.95
```

fluids.**motor_efficiency_underloaded**(*P*, *load=0.5*)

Returns the efficiency of a motor opperating under its design power according to *[R190]*.These values are generic; manufacturers usually list 4 points on their product information, but full-scale data is hard to find and not regulated.

Parameters **P** : float

Power, [W]

load : float, optional

Fraction of motor's rated electrical capacity being used

Returns **effciency** : float

Motor efficiency, [-]

Notes

If the efficiency returned by this function is unattractive, use a VFD. The curves used here are polynomial fits to *[R190]*'s graph, and curves were available for the following motor power ranges: 0-1 hp, 1.5-5 hp, 10 hp, 15-25 hp, 30-60 hp, 75-100 hp If above the upper limit of one range, the next value is returned.

References

[R190]

Examples

```
>>> motor_efficiency_underloaded(1*hp)
0.8705179600980149
>>> motor_efficiency_underloaded(10.1*hp,  .1)
0.6728425932357025
```

fluids.**Corripio_pump_efficiency**(*Q*)

Estimates pump efficiency using the method in Corripio (1982) as shown in *[R191]* and originally in *[R192]*. Estimation only

$$\eta_P = -0.316 + 0.24015\ln(Q) - 0.01199\ln(Q)^2$$

Parameters **Q** : float

Volumetric flow rate, [m^3/s]

Returns **effciency** : float

Pump efficiency, [-]

Notes

For Centrifugal pumps only. Range is 50 to 5000 GPM, but input variable is in metric. Values above this range and below this range will go negative, although small deviations are acceptable. Example 16.5 in *[R191]*.

References

[R191], *[R192]*

Examples

```
>>> Corripio_pump_efficiency(461./15850.323)
0.7058888670951621
```

fluids.**Corripio_motor_efficiency**(*P*)

Estimates motor efficiency using the method in Corripio (1982) as shown in *[R193]* and originally in *[R194]*. Estimation only.

$$\eta_M = 0.8 + 0.0319\ln(P_B) - 0.00182\ln(P_B)^2$$

Parameters **P** : float

Power, [W]

Returns **effciency** : float

Motor efficiency, [-]

Notes

Example 16.5 in *[R193]*.

References

[R193], *[R194]*

Examples

```
>>> Corripio_motor_efficiency(137*745.7)
0.9128920875679222
```

fluids.**specific_speed**(*Q*, *H*, *n=3600.0*)

Returns the specific speed of a pump operating at a specified Q, H, and n.

$$n_S = \frac{n\sqrt{Q}}{H^{0.75}}$$

Parameters **Q** : float

Flow rate, [m^3/s]

H : float

Head generated by the pump, [m]

n : float, optional

Speed of pump [rpm]

Returns **nS** : float

Specific Speed, [rpm*m^0.75/s^0.5]

Notes

Defined at the BEP, with maximum fitting diameter impeller, at a given rotational speed.

References

[R195]

Examples

Example from *[R195]*.

```
>>> specific_speed(0.0402, 100, 3550)
22.50823182748925
```

fluids.**specific_diameter**(*Q*, *H*, *D*)

Returns the specific diameter of a pump operating at a specified Q, H, and D.

$$D_s = \frac{DH^{1/4}}{\sqrt{Q}}$$

Parameters **Q** : float

Flow rate, [m^3/s]

H : float

Head generated by the pump, [m]

D : float

Pump impeller diameter [m]

Returns **Ds** : float

Specific diameter, [m^0.25/s^0.5]

Notes

Used in certain pump sizing calculations.

References

[R196]

Examples

```
>>> specific_diameter(Q=0.1, H=10., D=0.1)
0.5623413251903491
```

fluids.**speed_synchronous**(*f*, *poles=2*, *phase=3*)

Returns the synchronous speed of a synchronous motor according to *[R197]*.

$$N_s = \frac{120f \cdot \text{phase}}{\text{poles}}$$

Parameters **f** : float

Line frequency, [Hz]

poles : int, optional

The number of poles of the motor

phase : int, optional

Line AC phase

Returns **Ns** : float

Speed of synchronous motor, [rpm]

Notes

Synchronous motors have no slip. Large synchronous motors are not self-starting.

References

[R197]

Examples

```
>>> speed_synchronous(50, poles=12)
1500.0
>>> speed_synchronous(60, phase=1)
3600.0
```

fluids.**motor_round_size**(*P*)

Rounds up the power for a motor to the nearest NEMA standard power. The returned power is always larger or equal to the input power.

Parameters **P** : float

Power, [W]

Returns **P_actual** : float

Actual power, equal to or larger than input [W]

Notes

An exception is raised if the power required is larger than any of the NEMA sizes. Larger motors are available, but are unstandardized.

References

[R198]

Examples

```
>>> [motor_round_size(i) for i in [.1*hp, .25*hp, 1E5, 3E5]]
[186.42496789556753, 186.42496789556753, 111854.98073734052, 335564.94221202156]
```

fluids.**API520_round_size**(*A*)

Rounds up the area from an API 520 calculation to an API526 standard valve area. The returned area is always larger or equal to the input area.

Parameters **A** : float

Minimum discharge area [m^2]

Returns **area** : float

Actual discharge area [m^2]

Notes

To obtain the letter designation of an input area, lookup the area with the following:

API526_letters[API526_A.index(area)]

An exception is raised if the required relief area is larger than any of the API 526 sizes.

References

[R199]

Examples

From *[R199]*, checked with many points on Table 8.

```
>>> API520_round_size(1E-4)
0.00012645136
>>> API526_letters[API526_A.index(API520_round_size(1E-4))]
'E'
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

[R200] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R201] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R202] API. 2014. API 520 - Part 1 Sizing, Selection, and Installation of Pressure-relieving Devices, Part I - Sizing and Selection, 9E.

[R203] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R204] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R205] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R206] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R207] ISA. “RP75.23 Considerations for Evaluating Control Valve Cavitation.” 1995.

[R208] IEC 60534-2-1 / ISA-75.01.01-2007

[R209] IEC 60534-2-1 / ISA-75.01.01-2007

[R210] Green, Don, and Robert Perry. Perry’s Chemical Engineers’ Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R211] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R212] Green, Don, and Robert Perry. Perry’s Chemical Engineers’ Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R213] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R214] Gesellschaft, V. D. I., ed. VDI Heat Atlas. 2nd edition. Berlin; New York:: Springer, 2010.

[R215] Green, Don, and Robert Perry. Perry’s Chemical Engineers’ Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R216] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R217] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R218] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R219] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R220] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R221] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R222] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R223] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R224] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R225] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R226] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R227] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R228] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R229] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R230] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R231] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R232] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R233] Gesellschaft, V. D. I., ed. VDI Heat Atlas. 2nd edition. Berlin; New York:: Springer, 2010.

[R234] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R235] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R236] Gesellschaft, V. D. I., ed. VDI Heat Atlas. 2nd edition. Berlin; New York:: Springer, 2010.

[R237] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R238] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R239] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R240] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R241] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R242] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R243] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R244] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R245] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R246] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R247] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R248] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R248] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R250] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R251] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R252] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R253] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R254] Goldstein, Richard J. ECKERT NUMBER. Thermopedia. Hemisphere, 2011. 10.1615/AtoZ.e.eckert_number

[R255] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R256] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R257] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R258] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R259] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R260] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R261] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R262] Kundu, Pijush K., Ira M. Cohen, and David R. Dowling. Fluid Mechanics. Academic Press, 2012.

[R263] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R264] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R265] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R266] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R267] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R268] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R269] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R270] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R271] Haynes, W.M., Thomas J. Bruno, and David R. Lide. CRC Handbook of Chemistry and Physics. [Boca Raton, FL]: CRC press, 2014.

[R272] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R273] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R274] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R275] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R276] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R277] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R278] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R279] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R280] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R281] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R282] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R283] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R284] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R285] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R286] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R287] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R288] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R289] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R290] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R291] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R292] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R293] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R294] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R295] Rennels, Donald C., and Hobart M. Hudson. Pipe Flow: A Practical and Comprehensive Guide. 1st edition. Hoboken, N.J: Wiley, 2012.

[R296] Silverberg, Peter, and Ron Darby. "Correlate Pressure Drops through Fittings: Three Constants Accurately Calculate Flow through Elbows, Valves and Tees." Chemical Engineering 106, no. 7 (July 1999): 101.

[R297] Silverberg, Peter. "Correlate Pressure Drops Through Fittings." Chemical Engineering 108, no. 4 (April 2001): 127,129-130.

[R298] Hooper, W. B., "The 2-K Method Predicts Head Losses in Pipe Fittings," Chem. Eng., p. 97, Aug. 24 (1981).

[R299] Hooper, William B. "Calculate Head Loss Caused by Change in Pipe Size." Chemical Engineering 95, no. 16 (November 7, 1988): 89.

[R300] Kayode Coker. Ludwig's Applied Process Design for Chemical and Petrochemical Plants. 4E. Amsterdam; Boston: Gulf Professional Publishing, 2007.

[R301] ISA-75.01.01-2007 (60534-2-1 Mod) Draft

[R302] ISA-75.01.01-2007 (60534-2-1 Mod) Draft

[R303] ISA-75.01.01-2007 (60534-2-1 Mod) Draft

[R304] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R305] Moody, L.F.: An approximate formula for pipe friction factors. Trans. Am. Soc. Mech. Eng. 69,1005-1006 (1947)

[R306] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R307] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R308] Wood, D.J.: An Explicit Friction Factor Relationship, vol. 60. Civil Engineering American Society of Civil Engineers (1966)

[R309] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R310] Churchill, Stuart W. “Empirical Expressions for the Shear Stress in Turbulent Flow in Commercial Pipe.” AIChE Journal 19, no. 2 (March 1, 1973): 375-76. doi:10.1002/aic.690190228.

[R311] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R312] Eck, B.: Technische Stromungslehre. Springer, New York (1973)

[R313] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R314] Jain, Akalank K.”Accurate Explicit Equation for Friction Factor.” Journal of the Hydraulics Division 102, no. 5 (May 1976): 674-77.

[R315] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R316] Swamee, Prabhata K., and Akalank K. Jain.”Explicit Equations for Pipe-Flow Problems.” Journal of the Hydraulics Division 102, no. 5 (May 1976): 657-664.

[R317] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R318] Churchill, S.W.: Friction factor equation spans all fluid flow regimes. Chem. Eng. J. 91, 91-92 (1977)

[R319] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R320] Chen, Ning Hsing. “An Explicit Equation for Friction Factor in Pipe.” Industrial & Engineering Chemistry Fundamentals 18, no. 3 (August 1, 1979): 296-97. doi:10.1021/i160071a019.

[R321] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R322] Round, G. F.”An Explicit Approximation for the Friction Factor-Reynolds Number Relation for Rough and Smooth Pipes.” The Canadian Journal of Chemical Engineering 58, no. 1 (February 1, 1980): 122-23. doi:10.1002/cjce.5450580119.

[R323] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R324] Shacham, M. "Comments on: 'An Explicit Equation for Friction Factor in Pipe."' Industrial & Engineering Chemistry Fundamentals 19, no. 2 (May 1, 1980): 228-228. doi:10.1021/i160074a019.

[R325] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R326] Barr, Dih, and Colebrook White."Technical Note. Solutions Of The Colebrook-White Function For Resistance To Uniform Turbulent Flow." ICE Proceedings 71, no. 2 (January 6, 1981): 529-35. doi:10.1680/iicep.1981.1895.

[R327] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R328] Zigrang, D. J., and N. D. Sylvester."Explicit Approximations to the Solution of Colebrook's Friction Factor Equation." AIChE Journal 28, no. 3 (May 1, 1982): 514-15. doi:10.1002/aic.690280323.

[R329] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R330] Zigrang, D. J., and N. D. Sylvester."Explicit Approximations to the Solution of Colebrook's Friction Factor Equation." AIChE Journal 28, no. 3 (May 1, 1982): 514-15. doi:10.1002/aic.690280323.

[R331] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R332] Haaland, S. E."Simple and Explicit Formulas for the Friction Factor in Turbulent Pipe Flow." Journal of Fluids Engineering 105, no. 1 (March 1, 1983): 89-90. doi:10.1115/1.3240948.

[R333] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R334] Serghides T.K (1984)."Estimate friction factor accurately" Chemical Engineering, Vol. 91(5), pp. 63-64.

[R335] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R336] Serghides T.K (1984)."Estimate friction factor accurately" Chemical Engineering, Vol. 91(5), pp. 63-64.

[R337] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R338] Tsal, R.J.: Altshul-Tsal friction factor equation. Heat-Piping-Air Cond. 8, 30-45 (1989)

[R339] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R340] Manadilli, G.: Replace implicit equations with signomial functions. Chem. Eng. 104, 129 (1997)

[R341] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R342] Romeo, Eva, Carlos Royo, and Antonio Monzon."Improved Explicit Equations for Estimation of the Friction Factor in Rough and Smooth Pipes." Chemical Engineering Journal 86, no. 3 (April 28, 2002): 369-74. doi:10.1016/S1385-8947(01)00254-6.

[R343] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R344] Travis, Quentin B., and Larry W. Mays."Relationship between Hazen-William and Colebrook-White Roughness Values." Journal of Hydraulic Engineering 133, no. 11 (November 2007): 1270-73. doi:10.1061/(ASCE)0733-9429(2007)133:11(1270).

[R345] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R346] Rao, A.R., Kumar, B.: Friction factor for turbulent pipe flow. Division of Mechanical Sciences, Civil Engineering Indian Institute of Science Bangalore, India ID Code 9587 (2007)

[R347] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R348] Buzzelli, D.: Calculating friction in one step. Mach. Des. 80, 54-55 (2008)

[R349] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R350] Avci, Atakan, and Irfan Karagoz."A Novel Explicit Equation for Friction Factor in Smooth and Rough Pipes." Journal of Fluids Engineering 131, no. 6 (2009): 061203. doi:10.1115/1.3129132.

[R351] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R352] Papaevangelou, G., Evangelides, C., Tzimopoulos, C.: A New Explicit Relation for the Friction Factor Coefficient in the Darcy-Weisbach Equation, pp. 166-172. Protection and Restoration of the Environment Corfu, Greece: University of Ioannina Greece and Stevens Institute of Technology New Jersey (2010)

[R353] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R354] Brkic, Dejan."Review of Explicit Approximations to the Colebrook Relation for Flow Friction." Journal of Petroleum Science and Engineering 77, no. 1 (April 2011): 34-48. doi:10.1016/j.petrol.2011.02.006.

[R355] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R356] Brkic, Dejan."Review of Explicit Approximations to the Colebrook Relation for Flow Friction." Journal of Petroleum Science and Engineering 77, no. 1 (April 2011): 34-48. doi:10.1016/j.petrol.2011.02.006.

[R357] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R358] Fang, Xiande, Yu Xu, and Zhanru Zhou."New Correlations of Single-Phase Friction Factor for Turbulent Pipe Flow and Evaluation of Existing Single-Phase Friction Factor Correlations." Nuclear Engineering and Design, The International Conference on Structural Mechanics in Reactor Technology (SMiRT19) Special Section, 241, no. 3 (March 2011): 897-902. doi:10.1016/j.nucengdes.2010.12.019.

[R359] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R360] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R361] Rieger, F., V. Novak, and D. Havelkov (1988). The influence of the geometrical shape on the power requirements of ribbon impellers, Int. Chem. Eng., 28, 376-383.

[R362] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R363] Grenville, R. K., T. M. Hutchinson, and R. W. Higbee (2001). Optimisation of helical ribbon geometry for blending in the laminar regime, presented at MIXING XVIII, NAMF.

[R364] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R365] Giorges, Aklilu T. G., Larry J. Forney, and Xiaodong Wang. "Numerical Study of Multi-Jet Mixing." Chemical Engineering Research and Design, Fluid Flow, 79, no. 5 (July 2001): 515-22. doi:10.1205/02638760152424280.

[R366] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R367] Streiff, F. A., S. Jaffer, and G. Schneider (1999). Design and application of motionless mixer technology, Proc. ISMIP3, Osaka, pp. 107-114.

[R368] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R369] Streiff, F. A., S. Jaffer, and G. Schneider (1999). Design and application of motionless mixer technology, Proc. ISMIP3, Osaka, pp. 107-114.

[R370] Shen, John. "Discharge Characteristics of Triangular-Notch Thin-Plate Weirs: Studies of Flow to Water over Weirs and Dams." USGS Numbered Series. Water Supply Paper. U.S. Geological Survey: U.S. G.P.O., 1981

[R371] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R372] Kindsvater, Carl E., and Rolland W. Carter. "Discharge Characteristics of Rectangular Thin-Plate Weirs." Journal of the Hydraulics Division 83, no. 6 (December 1957): 1-36.

[R373] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R374] Normen für Wassermessungen: bei Durchführung von Abnahmeversuchen an Wasserkraftmaschinen. SIA, 1924.

[R375] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R376] Ackers, Peter, W. R. White, J. A. Perkins, and A. J. M. Harrison. Weirs and Flumes for Flow Measurement. Chichester; New York: John Wiley & Sons Ltd, 1978.

[R377] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R378] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R379] Normen für Wassermessungen: bei Durchführung von Abnahmeversuchen an Wasserkraftmaschinen. SIA, 1924.

[R380] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R382] King, H. W., Floyd A. Nagler, A. Streiff, R. L. Parshall, W. S. Pardoe, R. E. Ballester, Gardner S. Williams, Th Rehbock, Erik G. W. Lindquist, and Clemens Herschel. "Discussion of 'Precise Weir Measurements."' Transactions of the American Society of Civil Engineers 93, no. 1 (January 1929): 1111-78.

[R383] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R384] Kindsvater, Carl E., and Rolland W. Carter. "Discharge Characteristics of Rectangular Thin-Plate Weirs." Journal of the Hydraulics Division 83, no. 6 (December 1957): 1-36.

[R385] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R386] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R387] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R388] Chow, Ven Te. Open-Channel Hydraulics. New York: McGraw-Hill, 1959.

[R389] Chow, Ven Te. Open-Channel Hydraulics. New York: McGraw-Hill, 1959.

[R390] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R391] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R392] Chow, Ven Te. Open-Channel Hydraulics. New York: McGraw-Hill, 1959.

[R393] Ergun, S. (1952) 'Fluid flow through packed columns', Chem. Eng. Prog., 48, 89-94.

[R394] Kuo, K. K. and Nydegger, C., "Flow Resistance Measurement and Correlation in Packed Beds of WC 870 Ball Propellants," Journal of Ballistics , Vol. 2, No. 1, pp. 1-26, 1978.

[R395] Jones, D. P., and H. Krier. "Gas Flow Resistance Measurements Through Packed Beds at High Reynolds Numbers." Journal of Fluids Engineering 105, no. 2 (June 1, 1983): 168-172. doi:10.1115/1.3240959.

[R396] Jones, D. P., and H. Krier. "Gas Flow Resistance Measurements Through Packed Beds at High Reynolds Numbers." Journal of Fluids Engineering 105, no. 2 (June 1, 1983): 168-172. doi:10.1115/1.3240959.

[R397] P.C. Carman, Fluid flow through granular beds, Transactions of the London Institute of Chemical Engineers 15 (1937) 150-166.

[R398] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R399] Hicks, R. E. "Pressure Drop in Packed Beds of Spheres." Industrial Engineering Chemistry Fundamentals 9, no. 3 (August 1, 1970): 500-502. doi:10.1021/i160035a032.

[R400] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R401] H. Brauer, Grundlagen der Einphasen -und Mehrphasenstromungen, Sauerlander AG, Aarau, 1971.

[R402] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R403] Montillet, A., E. Akkari, and J. Comiti. "About a Correlating Equation for Predicting Pressure Drops through Packed Beds of Spheres in a Large Range of Reynolds Numbers." Chemical Engineering and Processing: Process Intensification 46, no. 4 (April 2007): 329-33. doi:10.1016/j.cep.2006.07.002.

[R404] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R405] Idelchik, I. E. Flow Resistance: A Design Guide for Engineers. Hemisphere Publishing Corporation, New York, 1989.

[R406] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R407] Benyahia, F., and K. E. O'Neill. "Enhanced Voidage Correlations for Packed Beds of Various Particle Shapes and Sizes." Particulate Science and Technology 23, no. 2 (April 1, 2005): 169-77. doi:10.1080/02726350590922242.

[R408] Benyahia, F., and K. E. O'Neill. "Enhanced Voidage Correlations for Packed Beds of Various Particle Shapes and Sizes." Particulate Science and Technology 23, no. 2 (April 1, 2005): 169-77. doi:10.1080/02726350590922242.

[R409] Benyahia, F., and K. E. O'Neill. "Enhanced Voidage Correlations for Packed Beds of Various Particle Shapes and Sizes." Particulate Science and Technology 23, no. 2 (April 1, 2005): 169-77. doi:10.1080/02726350590922242.

[R410] American National Standards Institute, and American Society of Mechanical Engineers. B36.10M-2004: Welded and Seamless Wrought Steel Pipe. New York: American Society of Mechanical Engineers, 2004.

[R411] American National Standards Institute, and American Society of Mechanical Engineers. B36-19M-2004: Stainless Steel Pipe. New York, N.Y.: American Society of Mechanical Engineers, 2004.

[R412] Oberg, Erik, Franklin D. Jones, and Henry H. Ryffel. Machinery's Handbook. Industrial Press, Incorporated, 2012.

[R413] Oberg, Erik, Franklin D. Jones, and Henry H. Ryffel. Machinery's Handbook. Industrial Press, Incorporated, 2012.

[R414] GoHz.com. Variable Frequency Drive Efficiency. http://www.variablefrequencydrive.org/vfd-efficiency

[R415] Natural Resources Canada. Electric Motors (1 to 500 HP/0.746 to 375 kW). As modified 2015-12-17. https://www.nrcan.gc.ca/energy/regulations-codes-standards/products/6885

[R416] Washington State Energy Office. Energy-Efficient Electric Motor Selection Handbook. 1993.

[R417] Seider, Warren D., J. D. Seader, and Daniel R. Lewin. Product and Process Design Principles: Synthesis, Analysis, and Evaluation. 2 edition. New York: Wiley, 2003.

[R418] Corripio, A.B., K.S. Chrien, and L.B. Evans, "Estimate Costs of Centrifugal Pumps and Electric Motors," Chem. Eng., 89, 115-118, February 22 (1982).

[R419] Seider, Warren D., J. D. Seader, and Daniel R. Lewin. Product and Process Design Principles: Synthesis, Analysis, and Evaluation. 2 edition. New York: Wiley, 2003.

[R420] Corripio, A.B., K.S. Chrien, and L.B. Evans, "Estimate Costs of Centrifugal Pumps and Electric Motors," Chem. Eng., 89, 115-118, February 22 (1982).

[R421] HI 1.3 Rotodynamic Centrifugal Pumps for Design and Applications

[R422] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R423] All About Circuits. Synchronous Motors. Chapter 13 - AC Motors http://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/synchronous-motors/

[R424] Natural Resources Canada. Electric Motors (1 to 500 HP/0.746 to 375 kW). As modified 2015-12-17. https://www.nrcan.gc.ca/energy/regulations-codes-standards/products/6885

[R425] API Standard 526.

[R1] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R2] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R3] API. 2014. API 520 - Part 1 Sizing, Selection, and Installation of Pressure-relieving Devices, Part I - Sizing and Selection, 9E.

[R4] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R5] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R6] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R7] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R8] ISA. "RP75.23 Considerations for Evaluating Control Valve Cavitation." 1995.

[R9] IEC 60534-2-1 / ISA-75.01.01-2007

[R10] IEC 60534-2-1 / ISA-75.01.01-2007

[R11] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R12] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R13] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R14] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R15] Gesellschaft, V. D. I., ed. VDI Heat Atlas. 2nd edition. Berlin; New York:: Springer, 2010.

[R16] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R17] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R18] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R19] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R20] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R21] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R22] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R23] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R24] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R25] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R26] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R27] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R28] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R29] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R30] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R31] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R32] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R33] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R34] Gesellschaft, V. D. I., ed. VDI Heat Atlas. 2nd edition. Berlin; New York:: Springer, 2010.

[R35] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R36] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R37] Gesellschaft, V. D. I., ed. VDI Heat Atlas. 2nd edition. Berlin; New York:: Springer, 2010.

[R38] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R39] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R40] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R41] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R42] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R43] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R44] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R45] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R46] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R47] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R48] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R49] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R49] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R51] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R52] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R53] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R54] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R55] Goldstein, Richard J. ECKERT NUMBER. Thermopedia. Hemisphere, 2011. 10.1615/AtoZ.e.eckert_number

[R56] Bergman, Theodore L., Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. Introduction to Heat Transfer. 6E. Hoboken, NJ: Wiley, 2011.

[R57] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R58] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R59] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R60] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R61] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R62] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R63] Kundu, Pijush K., Ira M. Cohen, and David R. Dowling. Fluid Mechanics. Academic Press, 2012.

[R64] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R65] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R66] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R67] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R68] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R69] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R70] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R71] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R72] Haynes, W.M., Thomas J. Bruno, and David R. Lide. CRC Handbook of Chemistry and Physics. [Boca Raton, FL]: CRC press, 2014.

[R73] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R74] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R75] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R76] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R77] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R78] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R79] Moody, L.F.: An approximate formula for pipe friction factors. Trans. Am. Soc. Mech. Eng. 69,1005-1006 (1947)

[R80] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R81] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R82] Wood, D.J.: An Explicit Friction Factor Relationship, vol. 60. Civil Engineering American Society of Civil Engineers (1966)

[R83] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R84] Churchill, Stuart W. "Empirical Expressions for the Shear Stress in Turbulent Flow in Commercial Pipe." AIChE Journal 19, no. 2 (March 1, 1973): 375-76. doi:10.1002/aic.690190228.

[R85] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R86] Eck, B.: Technische Stromungslehre. Springer, New York (1973)

[R87] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R88] Jain, Akalank K."Accurate Explicit Equation for Friction Factor." Journal of the Hydraulics Division 102, no. 5 (May 1976): 674-77.

[R89] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R90] Swamee, Prabhata K., and Akalank K. Jain."Explicit Equations for Pipe-Flow Problems." Journal of the Hydraulics Division 102, no. 5 (May 1976): 657-664.

[R91] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R92] Churchill, S.W.: Friction factor equation spans all fluid flow regimes. Chem. Eng. J. 91, 91-92 (1977)

[R93] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R94] Chen, Ning Hsing. “An Explicit Equation for Friction Factor in Pipe.” Industrial & Engineering Chemistry Fundamentals 18, no. 3 (August 1, 1979): 296-97. doi:10.1021/i160071a019.

[R95] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R96] Round, G. F.”An Explicit Approximation for the Friction Factor-Reynolds Number Relation for Rough and Smooth Pipes.” The Canadian Journal of Chemical Engineering 58, no. 1 (February 1, 1980): 122-23. doi:10.1002/cjce.5450580119.

[R97] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R98] Shacham, M. “Comments on: ‘An Explicit Equation for Friction Factor in Pipe.”’ Industrial & Engineering Chemistry Fundamentals 19, no. 2 (May 1, 1980): 228-228. doi:10.1021/i160074a019.

[R99] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R100] Barr, Dih, and Colebrook White.”Technical Note. Solutions Of The Colebrook-White Function For Resistance To Uniform Turbulent Flow.” ICE Proceedings 71, no. 2 (January 6, 1981): 529-35. doi:10.1680/iicep.1981.1895.

[R101] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R102] Zigrang, D. J., and N. D. Sylvester.”Explicit Approximations to the Solution of Colebrook’s Friction Factor Equation.” AIChE Journal 28, no. 3 (May 1, 1982): 514-15. doi:10.1002/aic.690280323.

[R103] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R104] Zigrang, D. J., and N. D. Sylvester.”Explicit Approximations to the Solution of Colebrook’s Friction Factor Equation.” AIChE Journal 28, no. 3 (May 1, 1982): 514-15. doi:10.1002/aic.690280323.

[R105] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R106] Haaland, S. E.”Simple and Explicit Formulas for the Friction Factor in Turbulent Pipe Flow.” Journal of Fluids Engineering 105, no. 1 (March 1, 1983): 89-90. doi:10.1115/1.3240948.

[R107] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R108] Serghides T.K (1984).”Estimate friction factor accurately” Chemical Engineering, Vol. 91(5), pp. 63-64.

[R109] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R110] Serghides T.K (1984).”Estimate friction factor accurately” Chemical Engineering, Vol. 91(5), pp. 63-64.

[R111] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R112] Tsal, R.J.: Altshul-Tsal friction factor equation. Heat-Piping-Air Cond. 8, 30-45 (1989)

[R113] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R114] Manadilli, G.: Replace implicit equations with signomial functions. Chem. Eng. 104, 129 (1997)

[R115] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R116] Romeo, Eva, Carlos Royo, and Antonio Monzon.”Improved Explicit Equations for Estimation of the Friction Factor in Rough and Smooth Pipes.” Chemical Engineering Journal 86, no. 3 (April 28, 2002): 369-74. doi:10.1016/S1385-8947(01)00254-6.

[R117] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R118] Travis, Quentin B., and Larry W. Mays.”Relationship between Hazen-William and Colebrook-White Roughness Values.” Journal of Hydraulic Engineering 133, no. 11 (November 2007): 1270-73. doi:10.1061/(ASCE)0733-9429(2007)133:11(1270).

[R119] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R120] Rao, A.R., Kumar, B.: Friction factor for turbulent pipe flow. Division of Mechanical Sciences, Civil Engineering Indian Institute of Science Bangalore, India ID Code 9587 (2007)

[R121] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R122] Buzzelli, D.: Calculating friction in one step. Mach. Des. 80, 54-55 (2008)

[R123] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R124] Avci, Atakan, and Irfan Karagoz.”A Novel Explicit Equation for Friction Factor in Smooth and Rough Pipes.” Journal of Fluids Engineering 131, no. 6 (2009): 061203. doi:10.1115/1.3129132.

[R125] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R126] Papaevangelou, G., Evangelides, C., Tzimopoulos, C.: A New Explicit Relation for the Friction Factor Coefficient in the Darcy-Weisbach Equation, pp. 166-172. Protection and Restoration of the Environment Corfu, Greece: University of Ioannina Greece and Stevens Institute of Technology New Jersey (2010)

[R127] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R128] Brkic, Dejan.”Review of Explicit Approximations to the Colebrook Relation for Flow Friction.” Journal of Petroleum Science and Engineering 77, no. 1 (April 2011): 34-48. doi:10.1016/j.petrol.2011.02.006.

[R129] Winning, H. and T. Coole. “Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes.” Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R130] Brkic, Dejan."Review of Explicit Approximations to the Colebrook Relation for Flow Friction." Journal of Petroleum Science and Engineering 77, no. 1 (April 2011): 34-48. doi:10.1016/j.petrol.2011.02.006.

[R131] Winning, H. and T. Coole. "Explicit Friction Factor Accuracy and Computational Efficiency for Turbulent Flow in Pipes." Flow, Turbulence and Combustion 90, no. 1 (January 1, 2013): 1-27. doi:10.1007/s10494-012-9419-7

[R132] Fang, Xiande, Yu Xu, and Zhanru Zhou."New Correlations of Single-Phase Friction Factor for Turbulent Pipe Flow and Evaluation of Existing Single-Phase Friction Factor Correlations." Nuclear Engineering and Design, The International Conference on Structural Mechanics in Reactor Technology (SMiRT19) Special Section, 241, no. 3 (March 2011): 897-902. doi:10.1016/j.nucengdes.2010.12.019.

[R133] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R134] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R135] Rieger, F., V. Novak, and D. Havelkov (1988). The influence of the geometrical shape on the power requirements of ribbon impellers, Int. Chem. Eng., 28, 376-383.

[R136] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R137] Grenville, R. K., T. M. Hutchinson, and R. W. Higbee (2001). Optimisation of helical ribbon geometry for blending in the laminar regime, presented at MIXING XVIII, NAMF.

[R138] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R139] Giorges, Aklilu T. G., Larry J. Forney, and Xiaodong Wang. "Numerical Study of Multi-Jet Mixing." Chemical Engineering Research and Design, Fluid Flow, 79, no. 5 (July 2001): 515-22. doi:10.1205/02638760152424280.

[R140] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R141] Streiff, F. A., S. Jaffer, and G. Schneider (1999). Design and application of motionless mixer technology, Proc. ISMIP3, Osaka, pp. 107-114.

[R142] Paul, Edward L, Victor A Atiemo-Obeng, and Suzanne M Kresta. Handbook of Industrial Mixing: Science and Practice. Hoboken, N.J.: Wiley-Interscience, 2004.

[R143] Streiff, F. A., S. Jaffer, and G. Schneider (1999). Design and application of motionless mixer technology, Proc. ISMIP3, Osaka, pp. 107-114.

[R144] Shen, John. "Discharge Characteristics of Triangular-Notch Thin-Plate Weirs: Studies of Flow to Water over Weirs and Dams." USGS Numbered Series. Water Supply Paper. U.S. Geological Survey: U.S. G.P.O., 1981

[R145] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R146] Kindsvater, Carl E., and Rolland W. Carter. "Discharge Characteristics of Rectangular Thin-Plate Weirs." Journal of the Hydraulics Division 83, no. 6 (December 1957): 1-36.

[R147] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R148] Normen für Wassermessungen: bei Durchführung von Abnahmeversuchen an Wasserkraftmaschinen. SIA, 1924.

[R149] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R150] Ackers, Peter, W. R. White, J. A. Perkins, and A. J. M. Harrison. Weirs and Flumes for Flow Measurement. Chichester; New York: John Wiley & Sons Ltd, 1978.

[R151] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R152] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R153] Normen für Wassermessungen: bei Durchführung von Abnahmeversuchen an Wasserkraftmaschinen. SIA, 1924.

[R154] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R156] King, H. W., Floyd A. Nagler, A. Streiff, R. L. Parshall, W. S. Pardoe, R. E. Ballester, Gardner S. Williams, Th Rehbock, Erik G. W. Lindquist, and Clemens Herschel. "Discussion of 'Precise Weir Measurements."' Transactions of the American Society of Civil Engineers 93, no. 1 (January 1929): 1111-78.

[R157] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R158] Kindsvater, Carl E., and Rolland W. Carter. "Discharge Characteristics of Rectangular Thin-Plate Weirs." Journal of the Hydraulics Division 83, no. 6 (December 1957): 1-36.

[R159] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R160] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R161] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R162] Chow, Ven Te. Open-Channel Hydraulics. New York: McGraw-Hill, 1959.

[R163] Chow, Ven Te. Open-Channel Hydraulics. New York: McGraw-Hill, 1959.

[R164] Blevins, Robert D. Applied Fluid Dynamics Handbook. New York, N.Y.: Van Nostrand Reinhold Co., 1984.

[R165] Cengel, Yunus, and John Cimbala. Fluid Mechanics: Fundamentals and Applications. Boston: McGraw Hill Higher Education, 2006.

[R166] Chow, Ven Te. Open-Channel Hydraulics. New York: McGraw-Hill, 1959.

[R167] Ergun, S. (1952) 'Fluid flow through packed columns', Chem. Eng. Prog., 48, 89-94.

[R168] Kuo, K. K. and Nydegger, C., "Flow Resistance Measurement and Correlation in Packed Beds of WC 870 Ball Propellants," Journal of Ballistics , Vol. 2, No. 1, pp. 1-26, 1978.

[R169] Jones, D. P., and H. Krier. "Gas Flow Resistance Measurements Through Packed Beds at High Reynolds Numbers." Journal of Fluids Engineering 105, no. 2 (June 1, 1983): 168-172. doi:10.1115/1.3240959.

[R170] Jones, D. P., and H. Krier. "Gas Flow Resistance Measurements Through Packed Beds at High Reynolds Numbers." Journal of Fluids Engineering 105, no. 2 (June 1, 1983): 168-172. doi:10.1115/1.3240959.

[R171] P.C. Carman, Fluid flow through granular beds, Transactions of the London Institute of Chemical Engineers 15 (1937) 150-166.

[R172] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R173] Hicks, R. E. "Pressure Drop in Packed Beds of Spheres." Industrial Engineering Chemistry Fundamentals 9, no. 3 (August 1, 1970): 500-502. doi:10.1021/i160035a032.

[R174] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R175] H. Brauer, Grundlagen der Einphasen -und Mehrphasenstromungen, Sauerlander AG, Aarau, 1971.

[R176] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R177] Montillet, A., E. Akkari, and J. Comiti. "About a Correlating Equation for Predicting Pressure Drops through Packed Beds of Spheres in a Large Range of Reynolds Numbers." Chemical Engineering and Processing: Process Intensification 46, no. 4 (April 2007): 329-33. doi:10.1016/j.cep.2006.07.002.

[R178] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R179] Idelchik, I. E. Flow Resistance: A Design Guide for Engineers. Hemisphere Publishing Corporation, New York, 1989.

[R180] Allen, K. G., T. W. von Backstrom, and D. G. Kroger. "Packed Bed Pressure Drop Dependence on Particle Shape, Size Distribution, Packing Arrangement and Roughness." Powder Technology 246 (September 2013): 590-600. doi:10.1016/j.powtec.2013.06.022.

[R181] Benyahia, F., and K. E. O'Neill. "Enhanced Voidage Correlations for Packed Beds of Various Particle Shapes and Sizes." Particulate Science and Technology 23, no. 2 (April 1, 2005): 169-77. doi:10.1080/02726350590922242.

[R182] Benyahia, F., and K. E. O'Neill. "Enhanced Voidage Correlations for Packed Beds of Various Particle Shapes and Sizes." Particulate Science and Technology 23, no. 2 (April 1, 2005): 169-77. doi:10.1080/02726350590922242.

[R183] Benyahia, F., and K. E. O'Neill. "Enhanced Voidage Correlations for Packed Beds of Various Particle Shapes and Sizes." Particulate Science and Technology 23, no. 2 (April 1, 2005): 169-77. doi:10.1080/02726350590922242.

[R184] American National Standards Institute, and American Society of Mechanical Engineers. B36.10M-2004: Welded and Seamless Wrought Steel Pipe. New York: American Society of Mechanical Engineers, 2004.

[R185] American National Standards Institute, and American Society of Mechanical Engineers. B36-19M-2004: Stainless Steel Pipe. New York, N.Y.: American Society of Mechanical Engineers, 2004.

[R186] Oberg, Erik, Franklin D. Jones, and Henry H. Ryffel. Machinery's Handbook. Industrial Press, Incorporated, 2012.

[R187] Oberg, Erik, Franklin D. Jones, and Henry H. Ryffel. Machinery's Handbook. Industrial Press, Incorporated, 2012.

[R188] GoHz.com. Variable Frequency Drive Efficiency. http://www.variablefrequencydrive.org/vfd-efficiency

[R189] Natural Resources Canada. Electric Motors (1 to 500 HP/0.746 to 375 kW). As modified 2015-12-17. https://www.nrcan.gc.ca/energy/regulations-codes-standards/products/6885

[R190] Washington State Energy Office. Energy-Efficient Electric Motor Selection Handbook. 1993.

[R191] Seider, Warren D., J. D. Seader, and Daniel R. Lewin. Product and Process Design Principles: Synthesis, Analysis, and Evaluation. 2 edition. New York: Wiley, 2003.

[R192] Corripio, A.B., K.S. Chrien, and L.B. Evans, "Estimate Costs of Centrifugal Pumps and Electric Motors," Chem. Eng., 89, 115-118, February 22 (1982).

[R193] Seider, Warren D., J. D. Seader, and Daniel R. Lewin. Product and Process Design Principles: Synthesis, Analysis, and Evaluation. 2 edition. New York: Wiley, 2003.

[R194] Corripio, A.B., K.S. Chrien, and L.B. Evans, "Estimate Costs of Centrifugal Pumps and Electric Motors," Chem. Eng., 89, 115-118, February 22 (1982).

[R195] HI 1.3 Rotodynamic Centrifugal Pumps for Design and Applications

[R196] Green, Don, and Robert Perry. Perry's Chemical Engineers' Handbook, Eighth Edition. McGraw-Hill Professional, 2007.

[R197] All About Circuits. Synchronous Motors. Chapter 13 - AC Motors http://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/synchronous-motors/

[R198] Natural Resources Canada. Electric Motors (1 to 500 HP/0.746 to 375 kW). As modified 2015-12-17. https://www.nrcan.gc.ca/energy/regulations-codes-standards/products/6885

[R199] API Standard 526.

f

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

V

W

Z