The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

April 28, 2019

Abstract

lua-physical is a pure Lua library which provides functions and object for doing computation with physical quantities. It has been written to simplify the creation problem sets. The package provides units of the SI and the imperial system. Furthermore an almost complete set of international currencies are supported, however without online exchange rates. In order to display the numbers with measurement uncertainties, the package is able to perform gaussian error propagation.

Contents

1 Introduction

The author of this package is a teacher at the high school *Kantonsschule Zug* in Switzerland. The main use of this package is to write physics problem sets. LuaLATEXdoes make it possible to integrate physical calculations directly. The package has been in use since 2016. Many bugs have been found and fixed. Nevertheless it still is possible, that some were not found yet. Therefore the author recommends not to use this package industry or science. If one does so, it's the responsability of the user to check results for plausability. If the user finds some bugs, they can be reported at github.com or directly to the author (`thomas.jenni(at)ksz.ch`).

2 Loading

This package is a pure Lua library. Therefore one has to require it explicitly by calling `require("physical")`. For printing results, the `siunitx` package can be used. It's recommended to define a macro like `\q` to convert the lua quantity object to a `siunitx` expression.
The following Latex preambel loads the `lua-physical` package and creates a macro `\q` for printing physical quantities.

Listing 1: basic preamble

```
\usepackage{lua-physical}
\usepackage{siunitx}

% configure siunitx
\sisetup{
  output-decimal-marker = {.},
  per-mode = symbol,
  separate-uncertainty = true,
  add-decimal-zero = true,
  exponent-product = \cdot,
  round-mode = off
}

% load lua-physical package
\begin{luacode*}
  physical = require("physical")
  N = physical.Number
\end{luacode*}

% print physical quantities
\newcommand{\q}[1]{%
  \directlua{tex.print(physical.Quantity.tosiunitx(#1,"scientific-notation=fixed,exponent-to-prefix=false"))}%
}
```

2.1 Dependencies

This is a standalone package. If a pretty print to LuaLaTeX is wanted, the package `siunitx` sould be installed.

3 Usage

Given the basic preamble, units can be used in lua code directly. By convention, all units have an underscore in front of them, i.e. Meter is `_m`, Second is `_s`. All available units are listed in chapter 4. The Result of the calculation can be printed to LuaLaTeX by using the macro `\q{}`.

Listing 2: The velocity of a car.

```
\begin{luacode}
  s = 10 * _m
  t = 2 * _s
  v = s/t
\end{luacode}

A car travels $\q{s}$ in $\q{t}$. calculate its velocity.
$$
  v=\frac{s}{t} = \frac{\q{s}}{\q{t}} = \q{v}
$$
```

A car travels 10 m in 2 s. Calculate its velocity.

$$v = \frac{s}{t} = \frac{10\,\mathrm{m}}{2\,\mathrm{s}} = 5\,\mathrm{m/s}$$

In the above listing 2, the variable `s` stands for displacement and has the unit meter `_m`. The variable `t` stands for time and is given in second `_s`. If mathematical operations are done on them, new physical quantities are created. In the problem above, the velocity `v` is calculated by dividing `s` by `t`. The instance `v` has the derived unit m/s. By using the macro `\q{}` all quantities can be printed to the LuaLaTeX code directly.

3.1 Unit conversion

Very often the result of a calculation is needed in different unit, than the given quantities are. In the following listing 3 the task is to calculate the volume of a cuboid with lengths given in different units. If the volume is calculated by multipling all three lengths, the unit of the result is cm mm m. If the unit cm^3 is preferred, it has to be converted explicitly. The conversion function is called `to()` and is available on all physical quantitiy instances. At first this looks a bit cumbersome. The reason of this behaviour is, that the software is not able to guess the unit of the result. In many cases, like in the example here, it's not clear what unit the result sould have. Therefore the user has always to give the target unit explicitly.

Listing 3: The volume of a cuboid.

```
\begin{luacode}
  a = 12 * _cm
  b = 150 * _mm
  c = 1.5 * _m

  V = a*b*c
\end{luacode}

Find the volume of a rectangular cuboid with lengths $\q{a}$,
$\q{b}$ and $\q{c}$.
$$
  V= a \cdot b \cdot c
  = \q{a} \cdot \q{b} \cdot \q{c}
  = \q{V}
  = \uuline{\q{V:to(_dm^3)}}
$$
```

Find the volume of a rectangular cuboid with lengths 12 cm, 150 mm and 1.5 m.

$$V = a \cdot b \cdot c = 12\,\mathrm{cm} \cdot 150\,\mathrm{mm} \cdot 1.5\,\mathrm{m} = 2700\,\mathrm{cm\,mm\,m} = \underline{\underline{27\,\mathrm{dm}^3}}$$

3.1.1 Temperature Conversion

In the following problem, listing 4 , the task is to convert a temperature given in the unit degree Celsius to Kelvin. As can be seen in the listing, the conversion function has two parameters.

The first argument is the target unit. The second is a boolean that tells the `to`-function to call a unit specific conversion function. By default the second argument is `false`.

Most units do not have a conversion function. Exceptions are the unit degree Celsius `_degC` and degree Fahrenheit `_degF`. These units are ambigous and can be interpreted as temperature differences or as an absolute temperatures. In the latter case, the conversion to base units is not a linear, but an affine transformation. This is because degree Celsius and degree Fahrenheit scales have their zero points at different temperatures compared to the unit Kelvin. Therefore these units have their own conversion functions.

By default `_degC` and `_degF` units are standing for temperature differences. If one wants to have it converted absolutely, the conversion function `to()` should have the second argument set to true.

Listing 4: Temperature conversion.

```
\begin{luacode}
```

```
  T = 20 * _degC
\end{luacode}

A thermometer shows $\q{T}$. Converte this quantity to Kelvin.
$$
  T = \q{T:to(_K)} + \q{273.15 * _K}
  =  \q{T:to(_K,true)}
$$
```

A thermometer shows 20 °C. Converte this quantity to Kelvin.

$$T = 20\,\mathrm{K} + 273.15\,\mathrm{K} = 293.15\,\mathrm{K}$$

3.1.2 Uncertainty

The package supports uncertainty propagation. To create a number with an uncertainty, an instance of `physical.Number` has to be created, see listing 5. It has to be remembered, that `N` is a alias for `physical.Number`. The first argument of the constructor `N(mean, uncertainty)` is the mean value and the second one the uncertainty of the measurement. If the proposed preamble is used, the uncertainty is by default seperated from the mean value by a plus-minus sign.
For the uncertainty propagation the gaussian formula

$$\Delta f = \sqrt{\left(\frac{\partial f}{x_1} \cdot \Delta x_1\right)^2 + \ldots + \left(\frac{\partial f}{x_n} \cdot \Delta x_2\right)^2}$$

is used. This formula is a good estimation for the uncertainty Δf, if the quantities $x_1, \ldots, x_n$ the function f depends on, have no correlation. Further, the function f has to behave linear, if the quantities x_i are changed in the range of their uncertainties.

Listing 5: Uncertainty in area calculation.

```
\begin{luacode}
  a = N(2,0.1) * _m
  b = N(3,0.1) * _m

  A = (a*b):to(_m^2)
\end{luacode}

Calculate the area of a rectangle with lengths $\q{a}$ and $\q{b}$.
$$
  A = a \cdot b
  = \q{a} \cdot \q{b}
  = \uuline{\q{A}}
$$
```

Calculate the area of a rectangle with lengths $(2.00 \pm 0.10)\,\mathrm{m}$ and $(3.00 \pm 0.10)\,\mathrm{m}$.

$$A = a \cdot b = (2.00 \pm 0.10)\,\mathrm{m} \cdot (3.00 \pm 0.10)\,\mathrm{m} = \underline{\underline{(6.0 \pm 0.4)\,\mathrm{m}^2}}$$

Instead of printing always the uncertainties, one can use the uncertainty calculation to provide significant numbers.
In the following problem, listing 6 , the task is to calculate the volume of an ideal gas. Given are pressure `p` in `_bar`, amount of substance `n` in `_mol` and temperature `T` in degree celsius `_degC`. In order to do the calculation, one has to convert `T`, which is given as an absolute temperature in degree celsius to the base unit Kelvin first. By setting `N.omitUncertainty = true`, all uncertainties are not printed.

Listing 6: Volume of an ideal gas.

```
\begin{luacode}
  N.omitUncertainty = true
  p = N(1.013,0.0001) * _bar
  n = N(1,0.01) * _mol
  T = N(30,0.1) * _degC

  V = ( n * _R * T:to(_K,true) / p ):to(_L)
\end{luacode}

An ideal gas ($\q{n}$) has a pressure of $\q{p}$ and a temperature of $\q{T}$. Calculate the volume of the gas.
$$
  V=\frac{ \q{n} \cdot \q{_R} \cdot \q{T:to(_K,true)} }{ \q{p} }
  = \q{V}
  = \uuline{\q{V}}
$$
```

An ideal gas ($1.0\,\mathrm{mol}$) has a pressure of $1.013\,\mathrm{bar}$ and a temperature of $30\,°\mathrm{C}$. Calculate the volume of the gas.

$$V = \frac{1.0\,\mathrm{mol} \cdot 8.31\,\mathrm{J/(mol\,K)} \cdot 303\,\mathrm{K}}{1.013\,\mathrm{bar}} = \underline{\underline{25\,\mathrm{L}}}$$

4 Supported Units

All supported units are listed in this chapter. Subchapter 4.1 lists the seven base units of the International System of Units (SI). In subchapter 4.2 mathematical and physical constants are defined. The subchapter 4.3 contains all derived units from the SI system. Subchapter 4.4 lists units, which are common but outside of the SI system. The subchapters 4.5 and 4.6 are dedicated to imperial and U.S. customary units. The last subchapter 4.7 containts international currencies.

4.1 Base Units

Quantity	Unit	Symbol	Dim.	Definition
time	second	`_s`	T	The SI unit of time. It is defined by taking the fixed numerical value of the caesium frequency $\Delta\nu_{Cs}$, the unperturbed ground-state hyperfine transition frequency of the caesium 133 atom, to be 9 192 631 770 when expressed in the unit 1/s.
length	meter	`_m`	L	The SI unit of length. It is defined by taking the fixed numercial value of the speed of light in vacuum c to be 299 792 458 when expressed in the unit of 1 m/s.
mass	kilogram	`_kg`	M	The SI unit of mass. It is defined by taking the fixed numerical value of the Planck constant h to be $6.626\,070\,15 \cdot 10^{-34}$ when expressed in $\mathrm{m}^2\,\mathrm{kg/s}$.
electric current	ampere	`_A`	I	The SI unit of electric current. It is defined by taking the fixed numerical value of the elementary charge e to be $1.602\,176\,634 \cdot 10^{-19}$ when expressed in A s.
thermodynamic temperature	kelvin	`_K`	Θ	The SI unit of the thermodynamic temperature. It is defineed by taking the fixed numerical value of the Boltzmann constant k_B to be $1.380\,649 \cdot 10^{-23}$ when expressed in $1\,\mathrm{kg\,m}^2/(\mathrm{s}^2\,\mathrm{K})$

Quantity	Unit	Symbol	Dim.	Definition
amount of substance	mole	`_mol`	N	The SI unit of amount of substance. One mole contains exactly $6.022\,140\,76 \cdot 10^{23}$ elementary entities. This number is the fixed numerical value of the Avogadro constant N_A when expressed in 1/mol.
luminous intensity	candela	`_cd`	J	The SI unit of luminous intensity in a given direction. It is defined by taking the fixed numerical value of the luminous efficacy of monochromatic radiation of frequency $5.4 \cdot 10^{14}\,\mathrm{Hz}$, K_{cd}, to be 683 when expressed in the unit $\mathrm{cd\,sr\,s^3/(kg\,m^2)}$.
number	–	`_1`	1	The dimensionless number one.
information	bit	`_bit`	B	The smallest amount of information.
currency	euro	`_EUR`	C	The value of the currency Euro.

Table 1: Base units of the International System of Units (SI) [3], information, currency and the dimensionless number one.

4.2 Constants

Almost all physical constants are taken from the NIST webpage [1]. The nominal values of solar, terrestrial and jovial quantities are taken from IAU Resolution B3 [4].

Name	Symbol	Dim.	Definition	Source
pi	_Pi	1	3.1415926535897932384626433832795028841971 * _1	
eulersnumber	_E	1	2.7182818284590452353602874713526624977572 * _1	
speedoflight	_c	$\mathrm{L}^{1}\,\mathrm{T}^{-1}$	299792458 * _m/_s	[1]
gravitationalconstant	_Gc	$\mathrm{L}^{3}\,\mathrm{M}^{-1}\,\mathrm{T}^{-2}$	N(6.67408e-11,3.1e-15) * _m^3/(_kg*_s^2)	[1]
planckconstant	_h_P	$\mathrm{L}^{2}\,\mathrm{M}^{1}\,\mathrm{T}^{-1}$	6.62607015e-34 * _J*_s	[1]
reducedplanckconstant	_h_Pbar	$\mathrm{L}^{2}\,\mathrm{M}^{1}\,\mathrm{T}^{-1}$	_h_P/(2*Pi)	[1]
elementarycharge	_e	$\mathrm{T}^{1}\,\mathrm{I}^{1}$	1.602176634e-19 * _C	[1]
vacuumpermeability	_u_0	$\mathrm{L}^{1}\,\mathrm{M}^{1}\,\mathrm{T}^{-2}\,\mathrm{I}^{-2}$	4e-7*Pi * _N/_A^2	[1]
vacuumpermitivity	_e_0	$\mathrm{L}^{-3}\,\mathrm{M}^{-1}\,\mathrm{T}^{4}\,\mathrm{I}^{2}$	1/(_u_0*_c^2)	[1]
atomicmassunit	_u	M^{1}	N(1.66053904e-27, 2e-35) * _kg	[1]
electronmass	_m_e	M^{1}	N(9.10938356e-31, 1.1e-38) * _kg	[1]
protonmass	_m_p	M^{1}	N(1.672621898e-27, 2.1e-35) * _kg	[1]
neutronmass	_m_n	M^{1}	N(1.674927471e-27, 2.1e-35) * _kg	[1]
bohrmagneton	_u_B	$\mathrm{L}^{2}\,\mathrm{I}^{1}$	_e*_h_Pbar/(2*_m_e)	[1]
nuclearmagneton	_u_N	$\mathrm{L}^{2}\,\mathrm{I}^{1}$	_e*_h_Pbar/(2*_m_p)	[1]

Name	Symbol	Dim.	Definition	Source
electronmagneticmoment	_u_e	$L^2 I^1$	N(-928.4764620e-26,5.7e-32) * _J/_T	[1]
protonmagneticmoment	_u_p	$L^2 I^1$	N(1.4106067873e-26,9.7e-35) * _J/_T	[1]
neutronmagneticmoment	_u_n	$L^2 I^1$	N(-0.96623650e-26,2.3e-26) * _J/_T	[1]
finestructureconstant	_alpha	1	_u_0*_e^2*_c/(2*_h_P)	[1]
rydbergconstant	_Ry	L^{-1}	_alpha^2*_m_e*_c/(2*_h_P)	[1]
avogadronumber	_N_A	N^{-1}	6.02214076e23/_mol	[1]
boltzmannconstant	_k_B	$L^2 M^1 T^{-2} \Theta^{-1}$	1.380649e-23 * _J/_K	[1]
molargasconstant	_R	$L^2 M^1 T^{-2} \Theta^{-1} N^{-1}$	N(8.3144598, 4.8e-6) * _J/(_K*_mol)	[1]
stefanboltzmannconstant	_sigma	$L^2 M^1 T^{-2} \Theta^{-1} N^{-1}$	Pi^2*_k_B^4/(60*_h_Pbar^3*_c^2)	[1]
standardgravity	_g_0	$L^1 T^{-2}$	9.80665 * _m/_s^2	[1]
nomsolradius	_R_S_nom	L^1	6.957e8 * _m	[4]
nomsolirradiance	_S_S_nom	$M^1 T^{-3}$	1361 * _W/_m^2	[4]
nomsolluminosity	_L_S_nom	$L^2 M^1 T^{-3}$	3.828e26 * _W	[4]
nomsolefftemperature	_T_S_nom	Θ^1	5772 * _K	[4]
nomsolmassparam	_GM_S_nom	$L^3 T^{-2}$	1.3271244e20 * _m^3 * _s^-2	[4]
nomterreqradius	_Re_E_nom	L^1	6.3781e6 * _m	[4]
nomterrpolradius	_Rp_E_nom	L^1	6.3568e6 * _m	[4]
nomterrmassparam	_GM_E_nom	$L^3 T^{-2}$	3.986004e14 * _m^3 * _s^-2	[4]

Name	Symbol	Dim.	Definition	Source
`nomjovianeqradius`	`_Re_J_nom`	L^1	`7.1492e7 * _m`	[4]
`nomjovianpolradius`	`_Rp_J_nom`	L^1	`6.6854e7 * _m`	[4]
`nomjovianmassparam`	`_GM_J_nom`	$\mathrm{L}^3\,\mathrm{T}^{-2}$	`1.2668653e17 * _m^3*_s^-2`	[4]

Table 2: Physical and mathematical constants.

4.3 Derived Units of the International System of Units (SI)

All units in this section are derived from the SI base units, [2, 123-129].

Quantity	Unit	Symbol	Dimension	Definition
frequency	hertz	`_Hz`	T^{-1}	`1/_s`
force	newton	`_N`	$\mathrm{M\,L\,T}^{-2}$	`_kg*_m/_s^2`
pressure	pascal	`_Pa`	$\mathrm{M\,L}^{-1}\,\mathrm{T}^{-2}$	`_N/_m^2`
energy	joule	`_J`	$\mathrm{M\,L}^{2}\,\mathrm{T}^{-2}$	`_N*_m`
power	watt	`_W`	$\mathrm{M\,L}^{2}\,\mathrm{T}^{-3}$	`_J/_s`
electric charge	coulomb	`_C`	$\mathrm{T\,I}$	`_A*_s`
electric potential difference	volt	`_V`	$\mathrm{M\,L}^{2}\,\mathrm{T}^{-3}\,\mathrm{I}^{-1}$	`_J/_C`
capacitance	farad	`_F`	$\mathrm{L}^{-2}\,\mathrm{M}^{-1}\,\mathrm{T}^{4}\,\mathrm{I}^{2}$	`_C/_V`
electric resistance	ohm	`_Ohm`	$\mathrm{L}^{2}\,M\mathrm{T}^{-3}\,\mathrm{I}^{-2}$	`_V/_A`
electric conductance	siemens	`_S`	$\mathrm{L}^{-2}\,\mathrm{M}^{-1}\,T^{3}I^{2}$	`_A/_V`
magnetic flux	weber	`_Wb`	$\mathrm{L}^{2}\,\mathrm{M\,T}^{-2}\,\mathrm{I}^{-1}$	`_V*_s`
magnetic flux density	tesla	`_T`	$\mathrm{M\,T}^{-2}\,\mathrm{I}^{-1}$	`_Wb/_m^2`
inductance	henry	`_H`	$\mathrm{L}^{2}\,\mathrm{M\,T}^{-2}\,\mathrm{I}^{-2}$	`_Wb/_A`
Celsius temperature	degree Celsius	`_degC`	Θ	`_K`
luminous flux	lumen	`_lm`	J	`_cd*_sr`
illuminance	lux	`_lux`	$\mathrm{L}^{-2}\,\mathrm{J}$	`_lm/_m^2`

Quantity	Unit	Symbol	Dimension	Definition
activity	becquerel	`_Bq`	T^{-1}	`1/_s`
absorbed dose	gray	`_Gy`	$\mathrm{L}^2\,\mathrm{T}^{-2}$	`_J/_kg`
dose equivalent	sievert	`_Sv`	$\mathrm{L}^2\,\mathrm{T}^{-2}$	`_J/_kg`
catalytic activity	katal	`_kat`	$\mathrm{T}^{-1}\,\mathrm{N}$	`_mol/_s`

Table 3: Derived units of the International System of Units (SI)

4.4 Units outside of the International System of Units (SI)

There are a few units with dimension 1. The unit Bel is only available with prefix decibel, because `_B` is the unit byte.

Quantity	Unit	Symbol	Dimension	Definition
length	angstrom	`_angstrom`	L	`1e-10*_m`
	fermi	`_fermi`	L	`1e-15*_m`
area	barn	`_barn`	L^2	`1e-28*_m^2`
	are	`_ar`	L^2	`1e2*_m^2`
	hectare	`_hectare`	L^2	`1e4*_m^2`
volume	liter	`_L`	L^3	`0.001*_m^3`
	metric teaspoon	`_tsp`	L^3	`0.005*_L`
	metric tablespoon	`_Tbsp`	L^3	`3*_tsp`
time	svedberg	`_svedberg`	T	`1e-13*_s`
	minute	`_min`	T	`_60*_s`
	hour	`_h`	T	`_60*_min`
	day	`_d`	T	`_24*_h`
	week	`_wk`	T	`_7*_d`
	year	`_a`	T	`365.25*_d`
plane angle	radian	`_rad`	1	`_1`
	degree	`_deg`	1	`(_Pi/180)*_rad`
	arc minute	`_arcmin`	1	`_deg/60`
	arc second	`_arcsec`	1	`_arcmin/60`
	gradian	`_gon`	1	`(Pi/200)*_rad`
	turn	`_tr`	1	`2*Pi*_rad`
solid angle	steradian	`_sr`	1	`_rad^2`
	spat	`_sp`	1	`4*Pi*_sr`
length	astronomical unit	`_au`	L	`149597870700*_m`
	lightyear	`_ly`	L	`_c*_a`
	lightsecond	`_ls`	L	`_c*_s`
	parsec	`_pc`	L	`(648000/_Pi)*_au`

Quantity	Unit	Symbol	Dimension	Definition
force	kilopond	`_kp`	$\mathrm{M\,L\,T^{-2}}$	`_kg*_g_0`
pressure	bar	`_bar`	$\mathrm{M\,L^{-1}\,T^{-2}}$	`1e5*_Pa`
	standardatmosphere	`_atm`	$\mathrm{M\,L^{-1}\,T^{-2}}$	`101325*_Pa`
	technicalatmosphere	`_at`	$\mathrm{M\,L^{-1}\,T^{-2}}$	`_kp/_cm^2`
	millimeterofmercury	`_mmHg`	$\mathrm{M\,L^{-1}\,T^{-2}}$	`133.322387415*_Pa`
	torr	`_Torr`	$\mathrm{M\,L^{-1}\,T^{-2}}$	`(101325/760)*_Pa`
energy	thermochemicalcalorie	`_cal`	$\mathrm{M\,L^{2}\,T^{-2}}$	`4.184*_J`
	internationalcalorie	`_cal_IT`	$\mathrm{M\,L^{2}\,T^{-2}}$	`4.1868*_J`
	gramoftnt	`_g_TNT`	$\mathrm{M\,L^{2}\,T^{-2}}$	`1e3 * _cal`
	tonoftnt	`_t_TNT`	$\mathrm{M\,L^{2}\,T^{-2}}$	`1e9 * _cal`
	electronvolt	`_eV`	$\mathrm{M\,L^{2}\,T^{-2}}$	`_e*_V`
	wattsecond	`_Ws`	$\mathrm{M\,L^{2}\,T^{-2}}$	`_W*_s`
	watthour	`_Wh`	$\mathrm{M\,L^{2}\,T^{-2}}$	`_W*_h`
power	voltampere	`_VA`	$\mathrm{M\,L^{2}\,T^{-3}}$	`_V*_A`
charge	amperesecond	`_As`	$\mathrm{T\,I}$	`_A*_s`
	amperehour	`_Ah`	$\mathrm{T\,I}$	`_A*_h`
information	nibble %	`_nibble`	B	`4*_bit`
	byte %	`_B`	B	`8*_bit`
information transfer rate	bitpersecond %	`_bps`	$\mathrm{T^{-1}\,B}$	`_bit/_s`

Quantity	Unit	Symbol	Dimension	Definition
number	percent %	`_percent`	1	`1e-2*_1`
	permille ‰	`_permille`	1	`1e-3*_1`
	parts-per-million	`_ppm`	1	`1e-6*_1`
	parts-per-billion	`_ppb`	1	`1e-9*_1`
	parts-per-trillion	`_ppt`	1	`1e-12*_1`
	parts-per-quadrillion	`_ppq`	1	`1e-15*_1`
	decibel	`_dB`	1	`_1`
mass	tonne	`_t`	M	`1000*_kg`

Table 4: Units outside of the International System of Units (SI)

4.5 Imperial Units

Quantity	Unit	Symbol	Dimension	Definition
length	inch	`_in`	L	`0.0254*_m`
	thou	`_th`	L	`0.001*_in`
	pica	`_pica`	L	`_in/6`
	point	`_pt`	L	`_in/72`
	hand	`_hh`	L	`4*_in`
	foot	`_ft`	L	`12*_in`
	yard	`_yd`	L	`3*_ft`
	rod	`_rd`	L	`5.5*_yd`
	chain	`_ch`	L	`4*_rd`
	furlong	`_fur`	L	`10*_ch`
	mile	`_mi`	L	`8*_fur`
	league	`_lea`	L	`3*_mi`
	nautical mile	`_nmi`	L	`1852 * _m`
	nautical league	`_nlea`	L	`3*_nmi`
	cable	`_cbl`	L	`_nmi/10`
	fathom	`_ftm`	L	`6*_ft`
velocity	knot	`_kn`	$L^1 T^{-1}$	`_nmi/_h`
area	acre	`_ac`	L^2	`43560*_ft^2`
volume	gallon	`_gal`	L^3	`4.54609*_L`
	quart	`_qt`	L^3	`_gal/4`
	pint	`_pint`	L^3	`_qt/2`
	cup	`_cup`	L^3	`_pint/2`
	gill	`_gi`	L^3	`_pint/4`
	fluid ounce	`_fl_oz`	L^3	`_gi/5`
	fluid dram	`_fl_dr`	L^3	`_fl_oz/8`

Quantity	Unit	Symbol	Dimension	Definition
mass	grain	`_gr`	M	`64.79891*_mg`
	pound	`_lb`	M	`7000*_gr`
	ounce	`_oz`	M	`_lb/16`
	dram	`_dr`	M	`_lb/256`
	stone	`_st`	M	`14*_lb`
	quarter	`_qtr`	M	`2*_st`
	hundredweight	`_cwt`	M	`4*_qtr`
	long ton	`_ton`	M	`20*_cwt`

Table 5: Imperial units

4.6 U.S. customary units

In the U.S., the length units are bound to the meter differently than in the imperial system. The followin definitions are taken from `https://en.wikipedia.org/wiki/United_States_customary_units`.

Quantity	Unit	Symbol	Dimension	Definition
length	U.S. survey inch	`_in_US`	L	`_m/39.37`
	U.S. survey hand	`_hh_US`	L	`4*_in_US`
	U.S. survey foot	`_ft_US`	L	`3*_hh_US`
	U.S. survey link	`_li_US`	L	`0.66*_ft_US`
	U.S. survey yard	`_yd_US`	L	`3*_ft_US`
	U.S. survey rod	`_rd_US`	L	`5.5*_yd_US`
	U.S. survey chain	`_ch_US`	L	`4*_rd_US`
	U.S. survey furlong	`_fur_US`	L	`10*_ch_US`
	U.S. survey mile	`_mi_US`	L	`8*_fur_US`
	U.S. survey league	`_lea_US`	L	`3*_mi_US`
	U.S. survey fathom	`_ftm_US`	L	`72*_in_US`
	U.S. survey cable	`_cbl_US`	L	`120*_ftm_US`
area	U.S. acre	`_ac_US`	L^2	`_ch_US*_fur_US`
volume	U.S. gallon	`_gal_US`	L^3	`231*_in^3`
	U.S. quart	`_qt_US`	L^3	`_gal_US/4`
	U.S. pint	`_pint_US`	L^3	`_qt_US/2`
	U.S. cup	`_pint_US`	L^3	`_pint_US/2`
	U.S. gill	`_gi_US`	L^3	`_pint_US/4`
	U.S. fluid ounce	`_fl_oz_US`	L^3	`_gi_US/4`
	U.S. table spoon	`_Tbsp_US`	L^3	`_fl_oz_US/2`
	U.S. tea spoon	`_tsp_US`	L^3	`_Tbsp_US/3`
	U.S. fluid dram	`_fl_dr_US`	L^3	`_fl_oz_US/8`

Quantity	Unit	Symbol	Dimension	Definition
mass	U.S. quarter	`_qtr_US`	L^3	`25*_lb`
	U.S. hundredweight	`_qtr_US`	L^3	`4*_qtr_US`
	U.S. short ton	`_ton_US`	L^3	`20*_cwt_US`

Table 6: U.S. customary units

4.7 International Currencies

Quantity	Unit	Symbol	Dimension	Definition
currency	Afghan afghani	_AFN	C	0.012*_EUR
	Albanian lek	_ALL	C	0.008*_EUR
	Armenian Dram	_AMD	C	0.0018*_EUR
	Angolan Kwanza	_AOA	C	0.0028*_EUR
	Argentine Peso	_ARS	C	0.021*_EUR
	Australian dollar	_AUD	C	0.63*_EUR
	Azerbaijani Manat	_AZN	C	0.63*_EUR
	Bosnian Mark	_BAM	C	0.51*_EUR
	Bangladeshi Taka	_BDT	C	0.011*_EUR
	Burundian Franc	_BIF	C	0.00049*_EUR
	Bolivian Boliviano	_BOB	C	0.13*_EUR
	Brazilian Real	_BRL	C	0.23*_EUR
	Botswana Pula	_BWP	C	0.083*_EUR
	Belarusian Ruble	_BYN	C	0.42*_EUR
	Canadian Dollar	_CAD	C	0.66*_EUR
	Congolese Franc	_CDF	C	0.00055*_EUR
	U.S. dollar	_USD	C	0.89*_EUR
	Japanese yen	_JPY	C	0.008*_EUR
	British pound	_GBP	C	1.17*_EUR
	Canadian dollar	_CAD	C	0.66*_EUR
	Swiss franc	_CHF	C	0.88*_EUR
	Chinese yuan	_CNY	C	0.13*_EUR
	Swedish krona	_SEK	C	0.094*_EUR
	New Zealand dollar	_NZD	C	0.60*_EUR

Table 7: Currency units based on exchange rates from 7.3.2019, 21:00 UTC.

5 Lua Documentation

In this chapter, the following shortcuts will be used.

```
local D = physical.Dimension
local U = physical.Unit
local N = physical.Number
local Q = physical.Quantity
```

The term `number` refers to a lua integer or a lua float number. By `string` a lua string is meant and by `bool` a lua boolean.

5.1 physical.Quantity

The quantity class is the main part of the library. Each physical Quantity and all units are represented by an instance of this class.

`Q.new(q=nil)`

Copy Constuctor

Parameters

`q` : `Q` or `number`, optional
: Optional argument is either `Q`, a `number` or `nil`.

`return` : `Q`
: The created `Q` instance

Note
As an argument it takes `Q`, `number` or `nil`. If `Q` is given, a copy of it is made and returned. If a `number` is given, the function creates a dimeensionless quantity with that value. In the case `nil` is given, the quantity `_1` is returned.

Example

```
myOne = Q()
myNumber = Q(42)
myLength = Q(73*_m)
```

`Q.defineBase(symbol,name,dimension)`

This function is used to declare the base units. Units are represented as `Q` instances.

Parameters

`symbol` : `string`
  symbol of the base quantity

`name` : `string`
  name of the base quantity

`dimension` : `D`
  Instance of the `D` class, which represents the dimension of the quantity.

`return` : `Q`
  The created `Q` instance.

Note

The function creates a global variable, an underscore concatenated with the `symbol` argument, e. g. `m` becomes the global variable `_m`.

The `name` is used for example in the siunitx conversion function, e.g `meter` will be converted to `\meter`.

Each quantity has a dimension associated with it. The argument `dimension` allows any dimension to be associated to base quantities. By default, the SI convention is used.

Example

```
Q.defineBase("m", "meter", L)
Q.defineBase("kg", "kilogram", M)
```

`Quantity.define(symbol, name, q, tobase=nil, frombase=nil)`

Creates a new derived unit from an expression of other units. For affine quantities like the temperature in celcius, one can give convertion functions to and from base units.

Parameters

`symbol` : `string`
  Symbol of the base quantity

`name` : `string`
  Name of the base quantity

`q` : `physical.Quantity`
  Definition of the unit

`tobase` : `function`, optional
  to convert a quantity to base units

`frombase` : `function`, optional
  to convert a quantity from the base units

`return` : `Quantity`
The defined quantity

Examples

```
Q.define("L", "liter", _dm^3)
Q.define("Pa", "pascal", _N/_m^2)
Q.define("C", "coulomb", _A*_s)

Q.define(
  "degC",
  "celsius",
  _K,
  function(q)
    q.value = q.value + 273.15
    return q
  end,
  function(q)
    q.value = q.value - 273.15
    return q
  end
)
```

`Quantity.definePrefix(symbol,name,factor)`

Defines a new prefix.

`symbol` : `string`, Symbol of the base quantity

`name` : `string`, Name of the base quantity

`factor` : `number`, the factor which corresponds to the prefix

```
Q.definePrefix("c", "centi", 1e-2)
Q.definePrefix("a", "atto", 1e-18)
```

`Quantity.addPrefix(prefixes, units)`

Create several units with prefixes from a given unit.

`prefixes` : `string`, list of unit symbols

`units` : `Quantity`, list of quantities

```
Q.addPrefix({"n","u","m","k","M","G"},{_m,_s,_A})
```

`Quantity.to(self,q,usefunction=false)`

Converts the quantity self to the unit of the quantity `q`. If the boolean `usefunction` is true, the convertion function is used for conversion.

`self` : `Quantity`

`q` : `Quantity`

`usefunction` : `Bool`

```
s = 1.9 * _km
print( s:to(_m) )
1900 * _m

T = 10 * _degC
print( T:to(_K) )
10 * _K
print( T:to(_K,true) )
283.15 * _K
```

`Quantity.tosiunitx(self,param,mode)`

Converts the quantity into a siunitx string.

`self` : `Quantity`

`param` : `string`

`mode` : `Number`, 0:\SI, 1:\num, 2:\si

```
s = 1.9 * _km

print( s:tosiunitx() )
\SI{1.9}{\kilo\meter}

print( s:tosiunitx(nil,1) )
\num{1.9}

print( s:tosiunitx(nil,2) )
\si{\kilo\meter}
```

`Quantity.isclose(self,q,r)`

Checks if this quantity is close to another one. The argument `r` is the maximal relative deviation.

`self` : `Quantity`

`q` : `Quantity,Number`

`r` : `Number`

```
s_1 = 1.9 * _m
s_2 = 2.0 * _m
print( s_1:isclose(s_2,0.1) )
true
print( s_1:isclose(s_2,0.01) )
false
```

`Quantity.min(q1, q2, ...)`

Returns the smallest quantity of several given ones. The function returns `q1` if the Quantities are equal.

`q1` : `Quantity,Number`, first argument

`q2` : `Quantity,Number`, second argument

```
s_1 = 15 * _m
s_2 = 5 * _m
print(s_1:min(s_2))
5 * _m
```

`Quantity.max(q1, q2, ...)`

Returns the biggest quantity of several given ones. The function returns `q1` if the Quantities are equal.

`q1` : `Quantity,Number`, first argument

`q2` : `Quantity,Number`, second argument

```
s_1 = 15 * _m
s_2 = 5 * _m
print(s_1:max(s_2))
15 * _m
```

`Quantity.abs(q)`

Returns the absolute value of the given quantity `q`.

`q` : `Quantity,Number`, argument

```
U = -5 * _V
print(U)
-5 * _V
print(U:abs())
5 * _V
```

`Quantity.sqrt(q)`

Returns the square root of the given quantity.

`q` : `Quantity,Number` argument

```
A = 25 * _m^2
s = sqrt(A)
print(s)

```

`Quantity.log(q, base)`

Returns the logarithm of the given quantitiy. If no base is given, the natural logarithm is calculated.

`q` : `Quantity,Number` dimensionless argument

`base` : `Quantity,Number` dimensionless argument

```
I = 1 * _W/_m^2
I_0 = 1e-12 * _W/_m^2
print(10 * (I/I_0):log(10) * _dB )
120 * _dB
```

`Quantity.exp(q)`

Returns the value of the exponential function of the given quantitiy.

`q` : `Quantity,Number` dimensionless argument

```
x = 2 * _1
print( x:exp() )
7.3890560989307
```

`Quantity.sin(q)`

Returns the value of the sinus function of the given quantitiy.

`q` : `Quantity,Number` dimensionless argument

```
alpha = 30 * _deg
print( alpha:sin() )
0.5
```

Quantity.cos(q)

Returns the value of the cosinus function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
alpha = 60 * _deg
print( alpha:cos() )
0.5
```

Quantity.tan(q)

Returns the value of the tangent function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
alpha = 45 * _deg
print( alpha:tan() )
1
```

Quantity.asin(q)

Returns the value of the arcus sinus function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
x = 0.5 * _1
print( x:asin():to(_deg) )
30 * _deg
```

Quantity.acos(q)

Returns the value of the arcus cosinus function of the given quantity. The quantity has to be dimensionless.

q : Quantity,Number dimensionless argument

```
x = 0.5 * _1
print( x:acos():to(_deg) )
60 * _deg
```

Quantity.atan(q)

Returns the value of the arcus tangent function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity`,`Number` dimensionless argument

```
x = 1 * _1
print( x:atan():to(_deg) )
45 * _deg
```

Quantity.sinh(q)

Returns the value of the hyperbolic sine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\sinh(x) = 0.5 \cdot e^x - 0.5/e^x \quad .$$

`q` : `Quantity`,`Number` dimensionless argument

```
x = 1 * _1
print( x:sinh() )
1.1752011936438
```

Quantity.cosh(q)

Returns the value of the hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\cosh(x) = 0.5 \cdot e^x + 0.5/e^x \quad .$$

`q` : `Quantity`,`Number` dimensionless argument

```
x = 1 * _1
print( x:cosh() )
1.5430806348152
```

`Quantity.tanh(q)`

Returns the value of the hyperbolic tangent function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad .$$

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:tanh() )
0.76159415595576
```

`Quantity.asinh(q)`

Returns the value of the inverse hyperbolic sine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{asinh}(x) = \ln\left(x + \sqrt{x^2 + 1}\right) \quad .$$

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:asinh() )
0.88137358701954
```

`Quantity.acosh(q)`

Returns the value of the inverse hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{acosh}(x) = \ln\left(x + \sqrt{x^2 - 1}\right) \quad , x > 1 \quad .$$

`q` : `Quantity,Number` dimensionless argument bigger than or equal to one.

```
x = 2 * _1
print( x:acosh() )
1.3169578969248
```

`Quantity.atanh(q)`

Returns the value of the inverse hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{atanh}(x) = \ln\left(\frac{1+x}{1-x}\right) \quad , -1 < x < 1 \quad .$$

`q` : `Quantity,Number` dimensionless argument with magnitude smaller than one.

```
x = 0.5 * _1
print( x:atanh() )
0.54930614433405
```

5.2 physical.Dimension

All physical quantities do have a physical dimension. For example the quantity *Area* has the dimension L^2 (lenght to the power of two). In the SI-System there are seven base dimensions, from which all other dimensions are derived. Each dimension is represented by an n-tuple, where n is the number of base dimensions. Each physical quantity has an associated dimension object. It is used two check if two quantities can be added or subtraced and if they are equal.

`Dimension.new(q=nil)`

Constructor of the `Dimension` class.

Parameters

`q` : `Dimension` or `string`, optional
: The name or symbol of the dimension. If `q` is a dimension, a copy of it is made. If no argument ist given, a dimension *zero* is created.

`return` : `Dimension`
: The created `Quantity` object

Notes

–

Examples

```
  V_1 = D("Velocity")
  L = D("L")
  V_2 = D(L/T)
```

5.3 physical.Unit

The task of this class is keeping track of the unit term. The unit term is a fraction of units. The units in the enumerator and denominator can have an exponent.

`Unit.new(u=nil)`

Copy Constructor. It copies a given unit object. If nothing is given, an empty unit is created.

Parameters

`u` : `Unit`
The unit object which will be copied.

`return` : `Unit`
The created `Unit` object

`Unit.new(symbol, name, prefixsymbol=nil, prefixname=nil)`

Constructor. A new `Unit` object with symbol is created. The prefixsymbol and prefixname are optional.

Parameters

`symbol` : `String`
The symbol of the unit.

`name` : `String`
The name of the unit.

`prefixsymbol` : `String`
The optional symbol of the prefix.

`prefixname` : `String`
The optional name of the prefix.

`return` : `Unit`
The created `Unit` object

`Unit.tosiunitx(self)`

The unit term will be compiled into a string, which the LaTeX package siunitx can understand.

Parameters

`return` : `String`
The siunitx representation of the unit term.

5.4 physical.Number

It does arithmetics with gaussian error propagation. A number instance has a mean value called `x` and an uncertainty called `dx`.

`Number.new(n=nil)`

This is the copy Constructor. It copies a given number object. If `n` is `nil`, an instance representing number zero with uncertainty zero is created.

Parameters

`n` : `Number`
The number object to be copied.

`return` : `Number`
The created `Number` instance.

`Number.new(x, dx)`

This constructor, creates a new instance with mean value `x` and uncertainty `dx`.

Parameters

`x` : `number`
mean value

`dx` : `number`
uncertainty value

`return` : `Number`
The created `Number` instance.

Examples

```
n = N(12,0.1)
print(n)
```

`Number.new(str)`

This constructor creates a new instance from a string. It can parse strings of the form `3.4`, `3.4e-3`, `5.4e-3 +/- 2.4e-6`, `5.45(7)e-23`.

Parameters / Return

`str` : `string`
The number as a string.

`return` : `Number`
The created `Number` object

Examples

```
n_1 = N("12.3e-6")
print(n_1)

n_2 = N("12 +/- 0.1")
print(n_2)

n_3 = N("12.0(1)")
print(n_3)
```

`Number.mean(n)`

Returns the mean value

Parameters / Return

`return` : `number`
The mean value

`Number.uncertainty(n)`

Returns the uncertainty value

Parameters / Return

`return` : `number`
The uncertainty value

`Number.abs(n)`

Returns the absolute value of the number.

Parameters / Return

`return` : `number`
The absolute value

`Number.sqrt(n)`

Returns the square root of the number.

Parameters / Return

`return` : `number`
The square root

References

[1] Webpage https://physics.nist.gov/cuu/index.html, August 2019.

[2] Bureau International des Poids et Mesures. The international system of units (si), 2006.

[3] Bureau International des Poids et Mesures. Resolutions of the 26th cgpm, November 2018.

[4] Prša et al. Nominal values for selected solar and planetary quantities: Iau 2015 resolution b3. The Astronomical Journal, 152:41, August 2016.