

The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

April 9, 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	3
2	Loading	3
2.1	Dependencies	4
3	Usage	4
3.1	Unit conversion	4
3.1.1	Temperature Conversion	5
3.1.2	Uncertainty	6
4	Supported Units	8
4.1	Base Units	8
4.2	Constants	10
4.3	Derived Units of the International System of Units (SI)	12
4.4	Units outside of the International System of Units (SI)	14
4.5	Imperial Units	16
4.6	U.S. customary units	18
4.7	International Currencies	20

5	Lua Documentation	21
5.1	physical.Quantity	21
5.2	physical.Dimension	30
5.3	physical.Unit	31
5.4	physical.Number	32

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. Lua_{La}T_EX does 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 responsibility 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](mailto:thomas.jenni@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

```
1  \usepackage{lua-physical}
2  \usepackage{siunitx}
3
4  % configure siunitx
5  \sisetup{
6    output-decimal-marker = {.},
7    per-mode = symbol,
8    separate-uncertainty = true,
9    add-decimal-zero = true,
10   exponent-product = \cdot,
11   round-mode = off
12 }
13
14 % load lua-physical package
15 \begin{luacode*}
16   physical = require("physical")
17   N = physical.Number
18 \end{luacode*}
19
20 % print physical quantities
21 \newcommand{\q}[1]{%
22   \directlua{tex.print(physical.Quantity.tosiunitx(#1,"scientific-
23     notation=fixed,exponent-to-prefix=false"))}%
24 }
```

2.1 Dependencies

This is a standalone package. If a pretty print to Lua^AT_EX is wanted, the package `siunitx` could 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 Lua^AT_EX by using the macro `\q{}`.

Listing 2: The velocity of a car.

```
1  \begin{luacode}
2      s = 10 * _m
3      t = 2 * _s
4      v = s/t
5  \end{luacode}
6
7  A car travels $\q{s}$ in $\q{t}$. calculate its velocity.
8  $$
9      v=\frac{s}{t} = \frac{\q{s}}{\q{t}} = \q{v}
10  $$
```

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

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

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 Lua^AT_EX 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 multiplying all three lengths, the unit of the result is `cm mm m`. If the unit `cm3` is preferred, it has to be converted explicitly. The conversion function is called `to()` and is available on all physical quantity 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 should have. Therefore the user has always to give the target unit explicitly.

Listing 3: The volume of a cuboid.

```

1  \begin{luacode}
2      a = 12 * _cm
3      b = 150 * _mm
4      c = 1.5 * _m
5
6      V = a*b*c
7  \end{luacode}
8
9  Find the volume of a rectangular cuboid with lengths  $\text{\textbackslash q{a}}\text{\$}$ ,
10  $\text{\textbackslash q{b}}\text{\$}$  and  $\text{\textbackslash q{c}}\text{\$}$ .
11  $\text{\textbackslash\textbackslash}$ 
12      V= a \cdot b \cdot c
13      = \text{\textbackslash q{a}} \cdot \text{\textbackslash q{b}} \cdot \text{\textbackslash q{c}}
14      = \text{\textbackslash q{V}}
15      = \underline{\text{\textbackslash q{V:to(_dm^3)}}}
16  $\text{\textbackslash\textbackslash}$ 

```

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

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

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 ambiguous 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.

```

1  \begin{luacode}

```

```

2   T = 20 * _degC
3   \end{luacode}
4
5   A thermometer shows  $\text{\q{T}}$ . Convert this quantity to Kelvin.
6   $$
7   T = \text{\q{T:to(_K)}} + \text{\q{273.15 * _K}}
8   = \text{\q{T:to(_K,true)}}
9   $$

```

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

$$T = 20 \text{ K} + 273.15 \text{ K} = 293.15 \text{ K}$$

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 separated from the mean value by a plus-minus sign.

For the uncertainty propagation the gaussian formula

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

is used. This formula is a good estimation for the uncertainty Δf , if the quantities x_1, \dots, 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.

```

1   \begin{luacode}
2     a = N(2,0.1) * _m
3     b = N(3,0.1) * _m
4
5     A = (a*b):to(_m^2)
6   \end{luacode}
7
8   Calculate the area of a rectangle with lengths  $\text{\q{a}}$  and  $\text{\q{b}}$ .
9   $$
10  A = a \cdot b
11  = \text{\q{a}} \cdot \text{\q{b}}
12  = \uuline{\q{A}}
13  $$

```

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

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

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.

```

1  \begin{luacode}
2      N.omitUncertainty = true
3      p = N(1.013,0.0001) * _bar
4      n = N(1,0.01) * _mol
5      T = N(30,0.1) * _degC
6
7      V = ( n * _R * T:to(_K,true) / p ):to(_L)
8  \end{luacode}
9
10 An ideal gas ( $\text{\textbackslash q{n}}$ ) has a pressure of  $\text{\textbackslash q{p}}$  and a temperature
    of  $\text{\textbackslash q{T}}$ . Calculate the volume of the gas.
11 $$
12 V=\frac{\text{\textbackslash q{n}} \cdot \text{\textbackslash q{R}} \cdot \text{\textbackslash q{T:to(_K,true)}}}{\text{\textbackslash q{p}}}
13 = \text{\textbackslash q{V}}
14 = \underline{\underline{\text{\textbackslash q{V}}}}
15 $$

```

An ideal gas (1.0 mol) has a pressure of 1.013 bar and a temperature of 30 °C. Calculate the volume of the gas.

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

4 Supported Units

In this chapter, all supported units are listed. Subchapter 4.1 lists the seven base units of the International System of Units (SI) and three additional ones. 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 is dedicated to imperial and U.S. customary units. The last subchapter 4.7 contains international currencies.

4.1 Base Units

Quantity	Unit	Symbol	Dim.	Definition
length	meter	<code>_m</code>	L	The distance light travels in vacuum during 1/299 792 458 second.
mass	kilogram	<code>_kg</code>	M	The mass of the international prototype of the kilogram.
time	second	<code>_s</code>	T	Is 9 192 631 770 times the period of the radiation from the transition between the two hyperfine levels of the ground state of caesium-133.
electric current	ampere	<code>_A</code>	I	The constant current which, if maintained in two straight parallel conductors of infinite length, of negligible circular cross-section, and placed 1 m apart in vacuum, would produce between these conductors a force equal to $2 \cdot 10^{-7}$ N/m.
thermodynamic temperature	kelvin	<code>_K</code>	Θ	Is the fraction 1/273.16 of the thermodynamic temperature of the triple point of water.
amount of substance	mole	<code>_mol</code>	N	Amount of substance that contains as many particles as there are atoms in 0.012 kg of carbon-12.
luminous intensity	candela	<code>_cd</code>	J	The luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequency $540 \cdot 10^{12}$ Hz and has a radiant intensity in that direction of (1/683) W/sr

Quantity	Unit	Symbol	Dim.	Definition
number	–	<code>_1</code>	1	The dimensionless number one.
information	bit	<code>_bit</code>	B	The smallest amount of information.
currency	euro	<code>_EUR</code>	C	The value of the currency Euro.

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

4.2 Constants

Name	Symbol	Dim.	Definition
pi	<code>_Pi</code>	1	$3.1415926535897932384626433832795028841971 * _1$
eulersnumber	<code>_E</code>	1	$2.7182818284590452353602874713526624977572 * _1$
speedoflight	<code>_c</code>	$L^1 T^{-1}$	$299792458 * _m/_s$
gravitationalconstant	<code>_Gc</code>	$L^3 M^{-1} T^{-2}$	$N(6.67408e-11, 3.1e-15) * _m^3/(_kg*s^2)$
planckconstant	<code>_h_P</code>	$L^2 M^1 T^{-1}$	$N(6.626070040e-34, 8.1e-42) * _J*s$
reducedplanckconstant	<code>_h_Pbar</code>	$L^2 M^1 T^{-1}$	$_h_P/(2*\Pi)$
elementarycharge	<code>_e</code>	$T^1 I^1$	$N(1.6021766208e-19, 9.8e-28) * _C$
vacuumpermeability	<code>_u_0</code>	$L^1 M^1 T^{-2} I^{-2}$	$4e-7*\Pi * _N/_A^2$
vacuumpermittivity	<code>_e_0</code>	$L^{-3} M^{-1} T^4 I^2$	$1/(_u_0*c^2)$
atomicmassunit	<code>_u</code>	M^1	$N(1.66053904e-27, 2e-35) * _kg$
electronmass	<code>_m_e</code>	M^1	$N(9.10938356e-31, 1.1e-38) * _kg$
protonmass	<code>_m_p</code>	M^1	$N(1.672621898e-27, 2.1e-35) * _kg$
neutronmass	<code>_m_n</code>	M^1	$N(1.674927471e-27, 2.1e-35) * _kg$
bohrmagneton	<code>_u_B</code>	$L^2 I^1$	$_e*_h_Pbar/(2*_m_e)$
nuclearmagneton	<code>_u_N</code>	$L^2 I^1$	$_e*_h_Pbar/(2*_m_p)$

Name	Symbol	Dim.	Definition
electronmagneticmoment	<code>_u_e</code>	$L^2 I^1$	$N(-928.4764620e-26, 5.7e-32) * _J/_T$
protonmagneticmoment	<code>_u_p</code>	$L^2 I^1$	$N(1.4106067873e-26, 9.7e-35) * _J/_T$
neutronmagneticmoment	<code>_u_n</code>	$L^2 I^1$	$N(-0.96623650e-26, 2.3e-26) * _J/_T$
finestructureconstant	<code>_alpha</code>	1	$_u_0 * _e^2 * _c / (2 * _h_P)$
rydbergconstant	<code>_Ry</code>	L^{-1}	$_alpha^2 * _m_e * _c / (2 * _h_P)$
avogadronumber	<code>_N_A</code>	N^{-1}	$N(6.022140857e23, 7.4e23) / _mol$
boltzmannconstant	<code>_k_B</code>	$L^2 M^1 T^{-2} \Theta^{-1}$	$N(1.38064852e-23, 7.9e-23) * _J/_K$
molargasconstant	<code>_R</code>	$L^2 M^1 T^{-2} \Theta^{-1} N^{-1}$	$N(8.3144598, 4.8e-6) * _J / (_K * _mol)$
stefanboltzmannconstant	<code>_sigma</code>	$L^2 M^1 T^{-2} \Theta^{-1} N^{-1}$	$P1^2 * _k_B^4 / (60 * _h_Pbar^3 * _c^2)$
standardgravity	<code>_g_0</code>	$L^1 T^{-2}$	$9.80665 * _m / _s^2$
nominalsolarradius	<code>_R_Sun</code>	L^1	$6.957e8 * _m$
nominalsolarirradiance	<code>_S_Sun</code>	$M^1 T^{-3}$	$1361 * _W / _m^2$
nominalsolarluminosity	<code>_L_Sun</code>	$L^2 M^1 T^{-3}$	$3.828e26 * _W$
nominalsolareffectivetemperature	<code>_T_Sun</code>	Θ^1	$5772 * _K$
nominalsolarmassparameter	<code>_GM_Sun</code>	$L^3 T^{-2}$	$1.3271244e20 * _m^3 * _s^{-2}$
nominalterrestrialequatorialradius	<code>_Re_E</code>	L^1	$6.3781e6 * _m$
nominalterrestrialpolarradius	<code>_Rp_E</code>	L^1	$6.3568e6 * _m$
nominalterrestrialmassparameter	<code>_GM_E</code>	$L^3 T^{-2}$	$3.986004e14 * _m^3 * _s^{-2}$
nominaljovianequatorialradius	<code>_Re_J</code>	L^1	$7.1492e7 * _m$
nominaljovianpolarradius	<code>_Rp_J</code>	L^1	$6.6854e7 * _m$
nominaljovianmassparameter	<code>_GM_J</code>	$L^3 T^{-2}$	$1.2668653e17 * _m^3 * _s^{-2}$

Table 2: Physical constants.

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

All units in this section are derived from the SI base units, [1, 123-129].

Quantity	Unit	Symbol	Dimension	Definition
plane angle	radian	<code>_rad</code>	1	<code>_1</code>
solid angle	steradian	<code>_sr</code>	1	<code>_rad^2</code>
frequency	hertz	<code>_Hz</code>	T^{-1}	<code>1/_s</code>
force	newton	<code>_N</code>	$M L T^{-2}$	<code>_kg*_m/_s^2</code>
pressure	pascal	<code>_Pa</code>	$M L^{-1} T^{-2}$	<code>_N/_m^2</code>
energy	joule	<code>_J</code>	$M L^2 T^{-2}$	<code>_N*_m</code>
power	watt	<code>_W</code>	$M L^2 T^{-3}$	<code>_J/_s</code>
electric charge	coulomb	<code>_C</code>	$T I$	<code>_A*_s</code>
electric potential difference	volt	<code>_V</code>	$M L^2 T^{-3} I^{-1}$	<code>_J/_C</code>
capacitance	farad	<code>_F</code>	$L^{-2} M^{-1} T^4 I^2$	<code>_C/_V</code>
electric resistance	ohm	<code>_Ohm</code>	$L^2 M T^{-3} I^{-2}$	<code>_V/_A</code>
electric conductance	siemens	<code>_S</code>	$L^{-2} M^{-1} T^3 I^2$	<code>_A/_V</code>
magnetic flux	weber	<code>_Wb</code>	$L^2 M T^{-2} I^{-1}$	<code>_V*_s</code>
magnetic flux density	tesla	<code>_T</code>	$M T^{-2} I^{-1}$	<code>_V*_s</code>
inductance	henry	<code>_H</code>	$L^2 M T^{-2} I^{-2}$	<code>_Wb/_A</code>
Celsius temperature	degree Celsius	<code>_degC</code>	Θ	<code>_K</code>
luminous flux	lumen	<code>_lm</code>	J	<code>_cd*_sr</code>
illuminance	lux	<code>_lux</code>	$L^{-2} J$	<code>_lm/_m^2</code>

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

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
	percent %	<code>_percent</code>	1	<code>1e-2*_1</code>
	permille ‰	<code>_permille</code>	1	<code>1e-3*_1</code>
	parts-per-million	<code>_ppm</code>	1	<code>1e-6*_1</code>
	parts-per-billion	<code>_ppb</code>	1	<code>1e-9*_1</code>
	parts-per-trillion	<code>_ppt</code>	1	<code>1e-12*_1</code>
	parts-per-quadrillion	<code>_ppq</code>	1	<code>1e-15*_1</code>
	decibel	<code>_dB</code>	1	<code>_1</code>
plane angle	degree	<code>_deg</code>	1	<code>(Pi/180)*_rad</code>
	arc minute	<code>_arcmin</code>	1	<code>_deg/60</code>
	arc second	<code>_arcsec</code>	1	<code>_arcmin/60</code>
	gradian	<code>_gon</code>	1	<code>(Pi/200)*_rad</code>
	turn	<code>_tr</code>	1	<code>2*Pi*_rad</code>
solid angle	spat	<code>_sp</code>	1	<code>4*Pi*_sr</code>
length	astronomical unit	<code>_au</code>	L	<code>149597870700*_m</code>
	lightyear	<code>_ly</code>	L	<code>_c*_a</code>
	parsec	<code>_pc</code>	L	<code>(648000/Pi)*_au</code>
	angstrom	<code>_angstrom</code>	L	<code>1e-10*_m</code>
	fermi	<code>_fermi</code>	L	<code>1e-15*_m</code>
area	are	<code>_ar</code>	L ²	<code>1e2*_m^2</code>
	hectare	<code>_hectare</code>	L ²	<code>1e4*_m^2</code>
	barn	<code>_barn</code>	L ²	<code>1e-28*_m^2</code>
volume	liter	<code>_L</code>	L ³	<code>0.001*_m^3</code>
	metric teaspoon	<code>_tsp</code>	L ³	<code>0.005*_L</code>
	metric tablespoon	<code>_Tbsp</code>	L ³	<code>3*_tsp</code>

Quantity	Unit	Symbol	Dimension	Definition
time	minute	<code>_min</code>	T	<code>_60*_s</code>
	hour	<code>_h</code>	T	<code>_60*_min</code>
	day	<code>_d</code>	T	<code>_24*_h</code>
	week	<code>_wk</code>	T	<code>_7*_d</code>
	year	<code>_a</code>	T	<code>365.25*_d</code>
	svedberg	<code>_svedberg</code>	T	<code>1e-13*_s</code>
mass	tonne	<code>_t</code>	M	<code>1000*_kg</code>

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

4.5 Imperial Units

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

Quantity	Unit	Symbol	Dimension	Definition
mass	grain	<code>_gr</code>	M	$64.79891*_\text{mg}$
	pound	<code>_lb</code>	M	$7000*_\text{gr}$
	ounce	<code>_oz</code>	M	$_\text{lb}/16$
	dram	<code>_dr</code>	M	$_\text{lb}/256$
	stone	<code>_st</code>	M	$14*_\text{lb}$
	quarter	<code>_qtr</code>	M	$2*_\text{st}$
	hundredweight	<code>_cwt</code>	M	$4*_\text{qtr}$
	long ton	<code>_ton</code>	M	$20*_\text{cwt}$

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

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

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
	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.

```
1 local D = physical.Dimension
2 local U = physical.Unit
3 local N = physical.Number
4 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 Constructor

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 dimensionless quantity with that value. In the case **nil** is given, the quantity **_1** is returned.

Example

```
1 myOne = Q()
2 myNumber = Q(42)
3 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

```

1 Q.defineBase("m", "meter", L)
2 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 conversion 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

```
1  Q.define("L", "liter", _dm^3)
2  Q.define("Pa", "pascal", _N/_m^2)
3  Q.define("C", "coulomb", _A*_s)
4
5  Q.define(
6      "degC",
7      "celsius",
8      _K,
9      function(q)
10         q.value = q.value + 273.15
11         return q
12     end,
13     function(q)
14         q.value = q.value - 273.15
15         return q
16     end
17 )
```

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

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

```
1  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 conversion function is used for conversion.

```
self : Quantity
q : Quantity
usefunction : Bool
```

```
1 s = 1.9 * _km
2 print( s:to(_m) )
3 1900 * _m
4
5 T = 10 * _degC
6 print( T:to(_K) )
7 10 * _K
8 print( T:to(_K,true) )
9 283.15 * _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
```

```
1 s = 1.9 * _km
2
3 print( s:tosiunitx() )
4 \SI{1.9}{\kilo\meter}
5
6 print( s:tosiunitx(nil,1) )
7 \num{1.9}
8
9 print( s:tosiunitx(nil,2) )
10 \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
```



```

1 s_1 = 1.9 * _m
2 s_2 = 2.0 * _m
3 print( s_1:isclose(s_2,0.1) )
4 true
5 print( s_1:isclose(s_2,0.01) )
6 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

```

1 s_1 = 15 * _m
2 s_2 = 5 * _m
3 print(s_1:min(s_2))
4 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

```

1 s_1 = 15 * _m
2 s_2 = 5 * _m
3 print(s_1:max(s_2))
4 15 * _m

```

Quantity.abs(q)

Returns the absolute value of the given quantity q.

q : Quantity,Number, argument

```

1 U = -5 * _V
2 print(U)
3 -5 * _V
4 print(U:abs())
5 5 * _V

```

Quantity.sqrt(q)

Returns the square root of the given quantity.

q : Quantity, Number argument

```
1 A = 25 * _m^2
2 s = sqrt(A)
3 print(s)
4
```

Quantity.log(q, base)

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

q : Quantity, Number dimensionless argument

base : Quantity, Number dimensionless argument

```
1 I = 1 * _W/_m^2
2 I_0 = 1e-12 * _W/_m^2
3 print(10 * (I/I_0):log(10) * _dB )
4 120 * _dB
```

Quantity.exp(q)

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

q : Quantity, Number dimensionless argument

```
1 x = 2 * _1
2 print( x:exp() )
3 7.3890560989307
```

Quantity.sin(q)

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

q : Quantity, Number dimensionless argument

```
1 alpha = 30 * _deg
2 print( alpha:sin() )
3 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`

```
1 alpha = 60 * _deg
2 print( alpha:cos() )
3 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`

```
1 alpha = 45 * _deg
2 print( alpha:tan() )
3 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`

```
1 x = 0.5 * _1
2 print( x:asin():to(_deg) )
3 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`

```
1 x = 0.5 * _1
2 print( x:acos():to(_deg) )
3 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

```
1 x = 1 * _1
2 print( x:atan():to(_deg) )
3 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

```
1 x = 1 * _1
2 print( x:sinh() )
3 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

```
1 x = 1 * _1
2 print( x:cosh() )
3 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

```
1  x = 1 * _1
2  print( x:tanh() )
3  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

```
1  x = 1 * _1
2  print( x:asinh() )
3  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.

```
1  x = 2 * _1
2  print( x:acosh() )
3  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.

```
1  x = 0.5 * _1
2  print( x:atanh() )
3  0.54930614433405
```

5.2 physical.Dimension

All physical quantities do have a physical dimension. For example the quantity *Area* has the dimension L^2 (length 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 to check if two quantities can be added or subtracted 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 is given, a dimension *zero* is created.

return: **Dimension**

The created **Quantity** object

Notes

—

Examples

```
1  V_1 = D("Velocity")
2  L = D("L")
3  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

```
1  n = N(12,0.1)
2  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

```
1  n_1 = N("12.3e-6")
2  print(n_1)
3
4  n_2 = N("12 +/- 0.1")
5  print(n_2)
6
7  n_3 = N("12.0(1)")
8  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] Bureau International des Poids et Mesures. The international system of units (si), 2006.