The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

February 15, 2019

Abstract

`lua-physical` is a pure Lua library which provides functions and object for doing computation with physical quantities. This package provides a standard set of units of the SI and the imperial system. It is possible to give a number a mesurement uncertainty.

is also integrated and is calculated by gaussian error propagation. The package includes some

Contents

1 Introduction **2**

2 Basic usage **2**

3 Supported Units **4**

4 Lua Documentation **9**

4.1 physical.Dimension 9

4.2 physical.Quantity 9

1 Introduction

The author of this package is a teacher at the *Kantonsschule Zug, Switzerland*, a high-school. The main use of this package is to write physics problem sets and integrate the calculation directly into the luatex-file. The package is now in use for more than two years and a lot of bugs have been found and crushed. Nevertheless it could be possible that some bugs are still there, living uncovered. Therefore I recommend not to use this library productively in 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, please report them on github.com or directly to the author.

E-Mail: `thomas.jenni(at)ksz.ch`

2 Basic usage

Since this package is pure lua library one has to require it explicitly by calling `require("physical")`. For printing results the `siunitx` package is used. It's recommended to define a shortcut like `\q` or `\Qty` to convert the lua quantity object to a siunitx expression. An example preamble is shown in the following.

```
\usepackage{siunitx}

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

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

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

Listing 1: basic preamble

Given the preamble one can use now units in lua code and insert results in the latex code.

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

A car travels $\q{s}$ in $\q{t}$. calculate its velocity.
$$
  v=\frac{s}{t} = \frac{\q{s}}{\q{t}} = \q{v} = \q{v:to(_km/_h)}
$$
```

Listing 2: basic usage

A car travels $10\,\mathrm{m}$ in $2\,\mathrm{s}$. Calculate its velocity.

$$v = \frac{s}{t} = \frac{10\,\mathrm{m}}{2\,\mathrm{s}} = 5\,\mathrm{m/s} = 18\,\mathrm{km/h}$$

3 Supported Units

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

Unit	Symbol	Definition
number	`_1`	The number one.
percent %	`_percent`	`1e-2*_1`
permille ‰	`_permille`	`1e-3*_1`
parts-per-million	`_ppm`	`1e-6*_1`
parts-per-billion	`_ppb`	`1e-9*_1`
parts-per-trillion	`_ppt`	`1e-12*_1`
parts-per-quadrillion	`_ppq`	`1e-15*_1`
decibel	`_dB`	`_1`

Table 1: Dimensionless units

Quantity	Unit	Symbol	Dim.	Definition
length	meter	`_m`	L	The distance light travels in vacuum during 1/299 792 458 second.
mass	kilogram	`_kg`	M	The mass of the international protoype of the kilogram.
time	second	`_s`	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	`_A`	I	The constant current which, if maintained in two straight parallel conductors of infinte length, of negligible circular crosss-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	`_K`	Θ	Is the fraction 1/273.16 of the thermodynamic temperature of the triple point of water.
amount of substance	mole	`_mol`	N	Amount of substance that contains as many particles as there are atoms in 0.012 kg of carbon-12.
luminous intensity	candela	`_cd`	J	the luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequenc $540 \cdot 10^{12}$ Hz and has a radiant intensity in that direction of (1/683) W/sr

Table 2: Base units of the International System of Units (SI)

Quantity	Unit	Symbol	Dimension	Definition
plane angle	radian	`_rad`	1	`\1`
solid angle	steradian	`_sr`	1	`_rad^2`
frequency	hertz	`_Hz`	T^{-1}	`1/_s`
force	newton	`_N`	$\mathrm{M}\,\mathrm{L}\,\mathrm{T}^{-2}$	`_kg*_m/_s^2`
pressure	pascal	`_Pa`	$\mathrm{M}\,\mathrm{L}^{-1}\,\mathrm{T}^{-2}$	`_N/_m^2`
energy	joule	`_J`	$\mathrm{M}\,\mathrm{L}^{2}\,\mathrm{T}^{-2}$	`_N*_m`
power	watt	`_W`	$\mathrm{M}\,\mathrm{L}^{2}\,\mathrm{T}^{-3}$	`_J/_s`
electric charge	coulomb	`_C`	$\mathrm{T}\,\mathrm{I}$	`_A*_s`
electric potential difference	volt	`_V`	$\mathrm{M}\,\mathrm{L}^{2}\,\mathrm{T}^{-3}\,\mathrm{I}^{-1}$	`_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`	$\mathrm{L}^{2}\,\mathrm{M}\,\mathrm{T}^{-2}\,\mathrm{I}^{-1}$	`_V*_s`
magnetic flux density	tesla	`_T`	$\mathrm{M}\,\mathrm{T}^{-2}\,\mathrm{I}^{-1}$	`_V*_s`
inductance	henry	`_H`	$\mathrm{L}^{2}\,\mathrm{M}\,\mathrm{T}^{-2}\,\mathrm{I}^{-2}$	`_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`
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)

Quantity	Unit	Symbol	Dim.	Definition
plane angle	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	spat	`_sp`	1	`4*Pi*_sr`
length	astronomical unit	`_au`	L	`149597870700*_m`
	lightyear	`_ly`	L	`_c*_a`
	parsec	`_pc`	L	`(648000/Pi)*_au`
	angstrom	`_angstrom`	L	`1e-10*_m`
	fermi	`_fermi`	L	`1e-15*_m`
area	are	`_ar`	L^2	`1e2*_m^2`
	hectare	`_hectare`	L^2	`1e4*_m^2`
	barn	`_barn`	L^2	`1e-28*_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	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`
	svedberg	`_svedberg`	T	`1e-13*_s`
mass	tonne	`_t`	M	`1000*_kg`

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

Quantity	Unit	Symbol	Dim.	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`

Table 5: Imperial units

4 Lua Documentation

In the following chapter, these shortcuts will be used.

```
local D = physical.Dimension
local Q = physical.Quantity
```

4.1 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(o=nil)`

Constructor of the `Dimension` class.

Parameters

`o` : `Dimension` or `string`, optional
The name or symbol of the dimension. If `o` 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)
```

4.2 physical.Quantity

`Quantity.new(o=nil)`

Constuctor of the `Quantity` class.

Parameters

`o` : `Quantity` or `number`, optional
: Optional argument for copying a `Quantity` or to create a `number` quantity

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

Notes

It takes an optional `quantity` or `number` for the argument `q`. If the argument `q` is given, the new quantity is a copy of it. If no argument ist given, a quantity `_1` is created.

Examples

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

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

A unit system has some special units, called base units. From those base units all other units are derived. This function is used to declare the base units. Since in this library units are the same thing as quantities, one has to define base quantities (units).

Parameters

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

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

`dimension` : `Dimension`
: object which represents the base Dimension of the base quantity

`return` : The created physical.`Quantity` object

Notes

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. The argument `dimension` allwos any dimension to be associated to base quantities.

Examples

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

`Quantity.define(symbol, name, o, 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

`o` : `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.min(o1, o2)

Compares two quantities and returns the smaller one. The function returns `o1` if the two Quantities are equal.

`o1` : `Quantity,Number`, first argument

`o2` : `Quantity,Number`, second argument

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

Quantity.max(o1, o2)

Compares two quantities and returns the bigger one. The function returns `o1` if the two Quantities are equal.

`o1` : `Quantity,Number`, first argument

`o2` : `Quantity,Number`, second argument

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

`Quantity.abs(o)`

Calculates and returns the absolute value of the given quantity.

`o` : `Quantity,Number`

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

`Quantity.sqrt(o)`

Calculates and returns the square root of the given quantity.

`o` : `Quantity,Number`

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

.sqrt(q)
.log(q, base)
.exp(q)
.sin(q)
.cos(q)
.tan(q)
.asin(q)
.acos(q)
.atan(q)
.sinh(q)
.cosh(q)
.tanh(q)
.asinh(q)
.acosh(q)
.atanh(q)
:to(o, usefunction)
:tosiunitx(param)
:tosiunitxsi(param)
:tosiunitxnum(param)
:isclose(o, r)