Physical units in UFL

Part 1/2

Michal Habera, Andreas Zilian

Department of Engineering | University of Luxembourg

18th June 2025 | FEniCS 2025

UNIVERSITÉ DU LUXEMBOURG

Contents

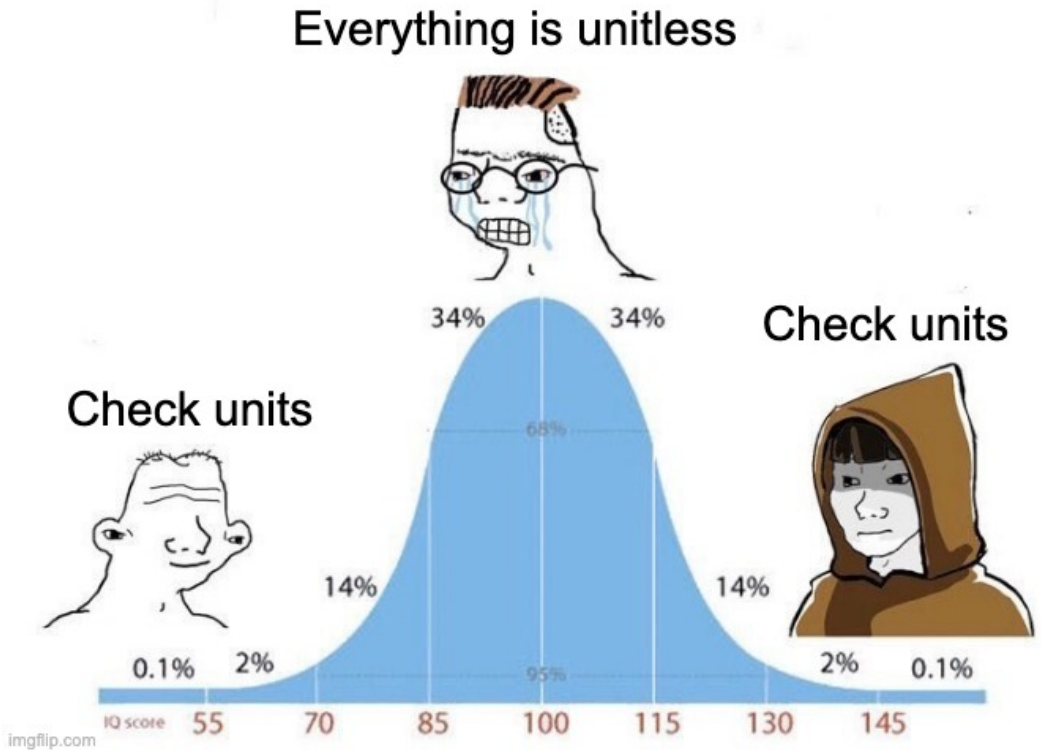
Disasters due to units errors

- **Discovery of American Continent (1492)**: Christopher Columbus unintentionally discovered America because during his travel preparations he mixed Arabic mile with a Roman mile which led to the wrong estimation of the equator and his expected travel distance.
- **Air Canada Flight 143 (1983)**: Ground crew of a Boeing 767 calculated required fuel in kilograms but loaded the aircraft using a meter calibrated in pounds per litre, resulting in approx. 10000 kg instead of 22300 kg. No injuries.
- **NASA Mars Clime Orbiter (1999)**: The navigation software that calculated accumulated thrust impulses expected newton-seconds (SI), but a subcontractor's module supplied pound-force-seconds (imperial). The spacecraft reached Mars 170 km lower than planned, disintegrated in the upper atmosphere, and wrote off 327 USD million in mission costs.

Unified Form Language with/out units?

Why should we care about units in UFL?

- UFL is a domain-specific language for finite element discretization of PDEs,
- PDEs often model a response of a physical system,
- physical systems are described by quantities with units.



Contents

Dimensions, units and a bit of theory

Quantity is a property of a material or system that can be quantified by measurement.

- Examples: length, time, temperature, electric current, force, energy, etc.,
- Quantities are often represented by symbols: l, t, Θ, I, F, W.

Dimension is a property of a quantity that is measurable and assigned to a specific phenomenon.

- Examples: length, time, temperature, electric current, force, energy, etc.,
- Dimensions are often represented by symbols: L, T, Θ, I, F, W.

Unit is a men-made reference scale (magnitude) for quantities that have a dimension.

- Examples: metre, second, kelvin, ampere, newton, joule, etc.,
- Units are also represented by symbols: m, s, K, A, N, J.

Dimensions, units and a bit of theory

Dimensional system is a set of dimensions that are used to describe a physical system.

- Examples: SI (Systeme International), CGS (Centimetre-Gram-Second), MKS (Metre-Kilogram-Second), etc.

Unit system is a dimensional system with a corresponding set of base units, i.e. a reference scale for each dimension.

- Examples: SI, metric, imperial, natural, etc.

Dimensions, units and a bit of theory

A dimensional system forms a **free commutative (abelian) group** with the operation of multiplication.
Any dimension $\mathbf{d}$ in a dimensional system $\mathbf{D}$ can be represented as a vector of rational numbers,

$$\mathbf{d} = (d_1, d_2, \ldots, d_n)^T \in \mathbb{Q}^n$$

where d_i is the exponent of the i-th dimension in the dimensional system $\mathbf{D}$.
In the vector space representation, the operation of multiplication is represented as a vector addition, i.e.

$$[ab] = [a][b] = \mathbf{d}_a + \mathbf{d}_b,$$

where $[a]$ and $[b]$ are the dimensions of the quantities a and b, respectively.

Example: in the MKS system, there is

$$[\text{velocity}] \simeq \mathbf{d}_{\text{velocity}} = (1, 0, -1)^T,$$
$$[\text{force}] \simeq \mathbf{d}_{\text{force}} = (1, 1, -2)^T.$$

SI and the seven units

SI and the seven units

International System of Units is a system of 7 base units and 22 derived units. Based on MKS system with most recent redefinition in 2019.
For a quantity q, we can write in the SI unit system

$$[q] = \mathsf{N}^{d_1}\mathsf{I}^{d_2}\mathsf{L}^{d_3}\mathsf{J}^{d_4}\mathsf{M}^{d_5}\Theta^{d_6}\mathsf{T}^{d_7} \tag{1}$$

where N, I, L, J, M, Θ, and T represent the seven SI base dimensions for amount of substance, electric current, length, luminous intensity, mass, temperature, and time, respectively. Their units are mole, ampere, metre, candela, kilogram, kelvin, and second.

Contents

Design decisions

Design goals for the implementation of the UFL unit system:

1. The implementation should not touch a single line of the UFL code (current UFL is in a messy state and requires serious cleanup).
2. The implementation should work as a testbed for future UFL-native solution, if community finds useful.
3. Use SymPy's Unit module as much as possible. Do not reinvent the wheel, use existing libraries for definition of unit symbols and conversions.
4. Unless specified otherwise, default to the SI unit system.
5. The implementation should allow for static dimensional analysis (sanity checks) and non-dimensionalization of UFL expressions.

Quantity

Quantity represent a physical quantity with a scale and a unit.

```
import sympy as sy
import sympy.physics.units as syu
from dolfiny.units import Quantity

rho = Quantity(mesh,
               scale=7.8,
               unit=syu.gram / syu.centimetre**3,
               symbol=sy.Symbol("rho"),
               unit_system=syu.si.SI)
```

The above reads like: ”rho” is a quantity with value of $7.8\,\mathrm{g}\cdot\mathrm{cm}^{-3}$.

Quantities inherit from the `ufl.Constant` and set the underlying numeric value of the Constant to the scale of the Quantity in the **base units** of the unit system.

1. Transformation

- UFL expressions are dimensionless by default, DAG nodes carry no unit information.
- No built-in unit consistency checks or dimensional analysis.
- `UnitTransformer` traverses DAG to add physical units.
- Replaces nodes with scaled expressions using `Quantity` objects according to a provided mapping.

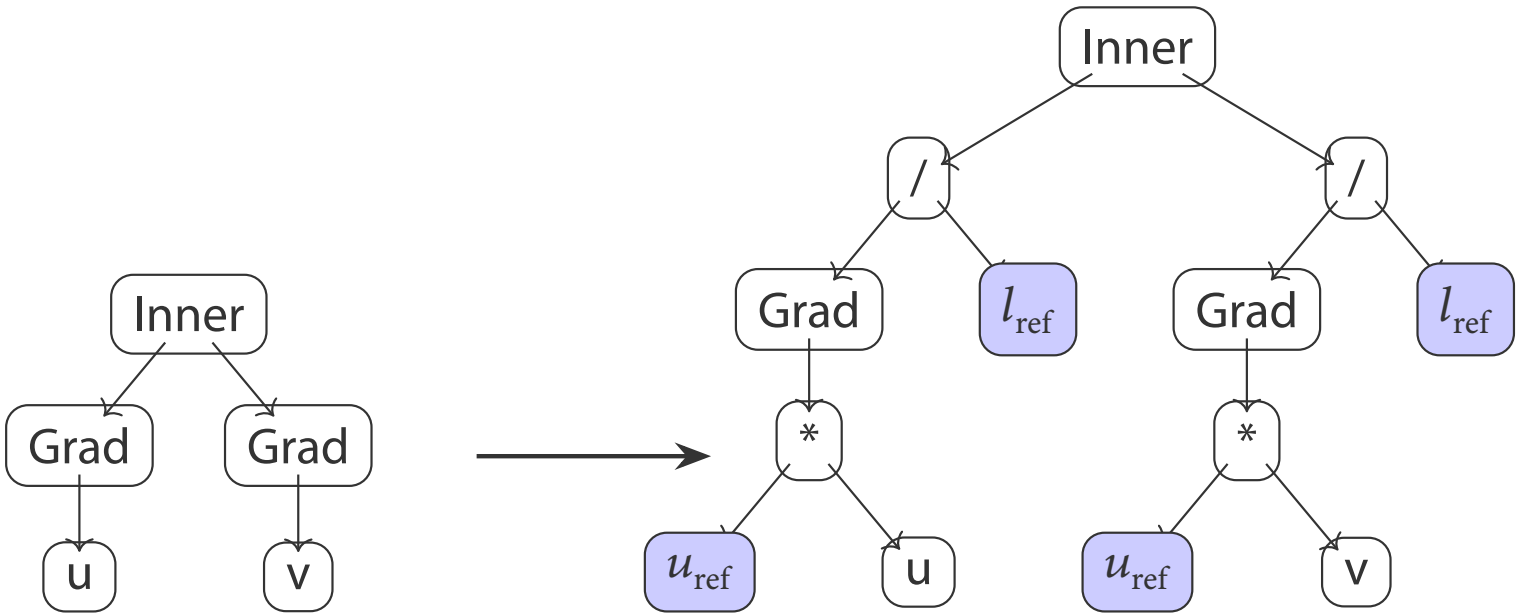
Example: Laplace (weak) operator $\nabla u \cdot \nabla v$. Consider the following mapping:

$$u \longrightarrow u_{\text{ref}}\, u, \qquad f \longrightarrow f_{\text{ref}}\, f,$$

$$v \longrightarrow u_{\text{ref}}\, v, \qquad \nabla \longrightarrow \frac{1}{l_{\text{ref}}} \nabla.$$

1. Transformation | Laplace operator example

```
expr = ufl.inner(ufl.grad(u), ufl.grad(v))
transfomed = dolfiny.units.transform(expr, mapping)
```



2. Factorization

Function $f : V \to W$ is called degree $k \in \mathbb{R}$ **positive homogeneous**, if

$$f(\alpha v) = \alpha^k f(v)$$

holds for all $\alpha \in \mathbb{F}, \alpha > 0$ all $v \in V$.

Lets assume that F is a UFL expression that satisfies

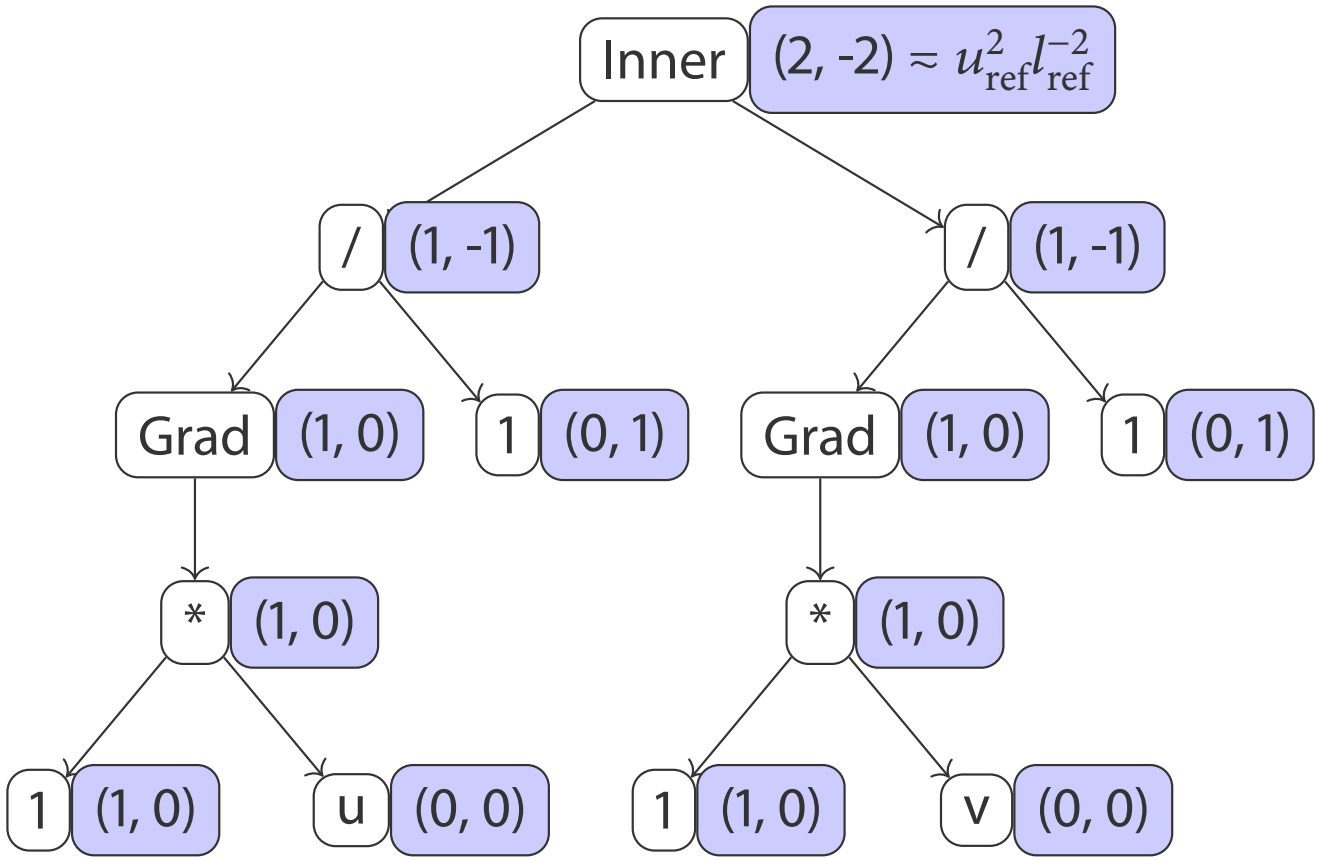
$$F(\alpha_1, \alpha_2, \ldots, \alpha_n) = \alpha_1^{k_1} \alpha_2^{k_2} \cdots \alpha_n^{k_n} F(1, 1, \ldots, 1)$$

for $\alpha_1, \alpha_2, \ldots, \alpha_n$ being `Quantity` objects.

The n-tuple $(k_1, k_2, \ldots, k_n)$ we call the **degree** of the expression F with respect to quantities $\alpha_1, \alpha_2, \ldots, \alpha_n$.

2. Factorization

```
quantities = [u_ref, l_ref]
factorized = dolfiny.units.factorize(expr, quantities, mapping)
```



2. Factorization

More generally, consider a sum node

$$F(x) = \alpha\, G(x) + \beta\, H(x)$$

which is non-homogeneous in x. Two failure modes occur:

1. *Inconsistent factors*: $\alpha \neq \beta$, and, as a special case,
2. *Inconsistent dimensions*: $[\alpha] \neq [\beta]$.

Contents

Examples | $H^1(\Omega)$ norm, inconsistent dimensions

```
u_ref = Quantity(mesh, 1.0, syu.meter, "u_ref")
l_ref = Quantity(mesh, 1.0, syu.meter, "l_ref")
quantities = [u_ref, l_ref]
mapping = {
    mesh.ufl_domain(): l_ref,
    u: u_ref * u,
    v: u_ref * v,
}
expr = ufl.inner(u, v) + ufl.inner(ufl.grad(u), ufl.grad(v))
factorized_expr = dolfiny.units.factorize(expr,
                                          quantities,
                                          mapping=mapping,
                                          mode="check")
```

Examples | $H^1(\Omega)$ norm, inconsistent dimensions

```
u_ref = Quantity(mesh, 1.0, syu.meter, "u_ref")
l_ref = Quantity(mesh, 1.0, syu.meter, "l_ref")
quantities = [u_ref, l_ref]
mapping = {
    mesh.ufl_domain(): l_ref,
    u: u_ref * u,
    v: u_ref * v,
}
expr = ufl.inner(u, v) + ufl.inner(ufl.grad(u), ufl.grad(v))
factorized_expr = dolfiny.units.factorize(expr,
                                          quantities,
                                          mapping=mapping,
                                          mode="check")
```

```
RuntimeError: Inconsistent dimensions
     u_ref * u * (conj((u_ref * v))).
---> +
     ({ A | A_{i_8} = (grad(u_ref * u))[i_8] / l_ref }) : ...
Different dimensions: Dimension(length**2) != Dimension(1).
```

Examples | $H^1(\Omega)$ norm, inconsistent factors

```
u_ref = Quantity(mesh, 1.0, syu.meter, "u_ref")
l_ref = Quantity(mesh, 1.0, syu.meter, "l_ref")
kappa = Quantity(mesh, 1.0, syu.meter, "kappa")
quantities = [u_ref, l_ref, kappa]
mapping = {
    mesh.ufl_domain(): l_ref,
    u: u_ref * u,
    v: u_ref * v,
}
expr = ufl.inner(u, v)
       + kappa ** 2 * ufl.inner(ufl.grad(u), ufl.grad(v))
factorized_expr = dolfiny.units.factorize(expr,
                                          quantities,
                                          mapping=mapping,
                                          mode="factorize")
```

Examples | $H^1(\Omega)$ norm, inconsistent factors

```
u_ref = Quantity(mesh, 1.0, syu.meter, "u_ref")
l_ref = Quantity(mesh, 1.0, syu.meter, "l_ref")
kappa = Quantity(mesh, 1.0, syu.meter, "kappa")
quantities = [u_ref, l_ref, kappa]
mapping = {
    mesh.ufl_domain(): l_ref,
    u: u_ref * u,
    v: u_ref * v,
}
expr = ufl.inner(u, v)
       + kappa ** 2 * ufl.inner(ufl.grad(u), ufl.grad(v))
factorized_expr = dolfiny.units.factorize(expr,
                                          quantities,
                                          mapping=mapping,
                                          mode="factorize")
```

```
RuntimeError: Inconsistent factors
     u_ref * u * (conj((u_ref * v)))
---> +
     kappa ** 2 * (({ A | A_{i_8} = (grad(u_ref * u))[i_8] ...
Different factors: u_ref**2 != kappa**2*u_ref**2/l_ref**2.
```

Weak Dirichlet BC imposition via a Nitsche method:

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx - \int_\Gamma \nabla u \cdot nv \, ds - \int_\Gamma \nabla v \cdot nu \, ds + \frac{\alpha}{h} \int_\Gamma uv \, ds.$$

```
u_ref = Quantity(mesh, 1.0, syu.volt, "u_ref")
l_ref = Quantity(mesh, 1.0, syu.meter, "l_ref")
alpha = Quantity(mesh, 10.0, sy.Integer(1), "alpha")

n = ufl.FacetNormal(mesh)
h = 2 * ufl.Circumradius(mesh)
a = ufl.inner(ufl.grad(u), ufl.grad(v)) * ufl.dx
a += -ufl.inner(n, ufl.grad(u)) * v * ufl.ds
a += -ufl.inner(n, ufl.grad(v)) * u * ufl.ds
a += alpha / h * ufl.inner(u, v) * ufl.ds

dim = dolfiny.units.get_dimension(a, quantities, mapping)
```

Examples | Stabilizations, penalty methods and friends

```
RuntimeError: Inconsistent dimensions across integrals in Form.
Scales: u_ref**2 != alpha*l_ref*u_ref**2.
Dimensions: Dimension(voltage**2) != Dimension(length*voltage**2).
```

```
RuntimeError: Inconsistent dimensions across integrals in Form.
Scales: u_ref**2 != alpha*l_ref*u_ref**2.
Dimensions: Dimension(voltage**2) != Dimension(length*voltage**2).
```

Solution:

```python
# ...
alpha = Quantity(mesh, 10.0, 1 / syu.meter, "alpha")
# ...
dimsys = syu.si.SI.get_dimension_system()
assert dimsys.equivalent_dims(dim, syu.voltage**2)
```

Summary

Physical units in UFL implementation allows you to:

- define dimensional quantities,
- extract and compare dimensions of expressions,
- perform homogeneous factorization.

Physical units in UFL implementation **does not** allow you to:

- introduce vector/tensor valued quantities,
- factorize vector/tensor expressions with different dimensions per component,
- apply non-linear transformations.

Summary

Physical units in UFL implementation allows you to:

- define dimensional quantities,
- extract and compare dimensions of expressions,
- perform homogeneous factorization.

Physical units in UFL implementation **does not** allow you to:

- introduce vector/tensor valued quantities,
- factorize vector/tensor expressions with different dimensions per component,
- apply non-linear transformations.

Cool Michal, but can I:

- derive dimensionless numbers?
- explore relative scaling of terms in UFL forms?

Summary

Physical units in UFL implementation allows you to:

- define dimensional quantities,
- extract and compare dimensions of expressions,
- perform homogeneous factorization.

Physical units in UFL implementation **does not** allow you to:

- introduce vector/tensor valued quantities,
- factorize vector/tensor expressions with different dimensions per component,
- apply non-linear transformations.

Cool Michal, but can I:

- derive dimensionless numbers?
- explore relative scaling of terms in UFL forms?

Yes, you can!

See Andreas' talk on Friday.


IT AIN'T MUCH
BUT IT'S HONEST WORK