Physlib Monthly Report

1–31 January 2026

Contributors: Adomas Baliuka, Joseph Tooby-Smith, Miles Cranmer, Nicola Bernini, Nikolai Kashcheev, physlean0, rht, Riad F, ValentinBredemestre

Reviewers: Joseph Tooby-Smith

Generated 14 August 2026 from commit `d127ae0`.

Abstract

Physlib is a community library of formalised physics for the Lean 4 proof assistant: definitions and results from across physics, stated in a formal language and checked by machine. This report is an automatically generated record of one month's additions to it.

It covers 1–31 January 2026 on branch `master` of `leanprover-community/physlib`, between commits `7ab2194` and `d127ae0`. Over that period 9 contributors landed 35 commits, changing 2,554 lines across 29 files and adding 104 new Lean declarations, each catalogued with its statement in Section 2. The complete diff is available at GitHub compare.

Contents

1 Introduction **2**
- 1.1 Contributors . . . 2
- 1.2 Reviewers . . . 2
- 1.3 Modified subfolders . . . 2
- 1.4 New declarations by kind . . . 3

2 Details by file **3**
- 2.1 `PhysLean.ClassicalMechanics.DampedHarmonicOscillator.Basic` . . . 3
- 2.2 `PhysLean.ClassicalMechanics.HarmonicOscillator.Solution` . . . 5
- 2.3 `PhysLean.ClassicalMechanics.Lagrangian.TotalDerivativeEquivalence` . . . 6
- 2.4 `PhysLean.CondensedMatter.TightBindingChain.Basic` . . . 7
- 2.5 `PhysLean.Particles.BeyondTheStandardModel.TwoHDM.Basic` . . . 8
- 2.6 `PhysLean.Particles.BeyondTheStandardModel.TwoHDM.GramMatrix` . . . 8
- 2.7 `PhysLean.Particles.BeyondTheStandardModel.TwoHDM.Potential` . . . 12
- 2.8 `PhysLean.Particles.StandardModel.Basic` . . . 16
- 2.9 `PhysLean.Particles.StandardModel.HiggsBoson.Basic` . . . 17
- 2.10 `PhysLean.Relativity.Tensors.RealTensor.ToComplex` . . . 17

1 Introduction

During Jan 2026, 9 contributors submitted 35 commits that changed 2,554 lines (+2,153 / -401) across 29 files spanning 15 top-level areas of the repository. Of those, 4 files were newly added and 2 were removed outright. This report catalogues 104 newly added Lean declarations: 72 lemmas, 26 defs, 3 structures, and 3 instances. We're delighted to recognize the following first-time contributors this month: Adomas Baliuka, Miles Cranmer, Nikolai Kashcheev, physlean0, Riad F, and ValentinBredemestre.

See the full compare diff and the list of commits on GitHub for the raw source.

1.1 Contributors

The following individuals authored commits merged during this reporting period, listed alphabetically.

Contributor	Lines Changed	Commits
Adomas Baliuka	3	1
Joseph Tooby-Smith	1,571	11
Miles Cranmer	2	1
Nicola Bernini	341	4
Nikolai Kashcheev	368	6
physlean0	99	1
rht	137	3
Riad F	67	2
ValentinBredemestre	48	6

1.2 Reviewers

The following individuals approved pull requests during this reporting period, listed alphabetically.

Reviewer	PRs Reviewed
Joseph Tooby-Smith	24

1.3 Modified subfolders

Subfolder	Files	New	Deleted	Additions	Deletions
`PhysLean/Particles`	7	+1	-1	+1,077	-296
`PhysLean/ClassicalMechanics`	4	+2	–	+727	-12
`PhysLean/Relativity`	4	–	–	+133	-33
`PhysLean/CondensedMatter`	1	–	–	+94	-5
`.github/ISSUE_TEMPLATE`	2	+1	-1	+33	-24
`PhysLean/QFT`	1	–	–	+40	-9
`README.md`	1	–	–	+22	-8
`.github/workflows`	1	–	–	+10	-0
`PhysLean/StatisticalMechanics`	1	–	–	+5	-4
`PhysLean/QuantumMechanics`	2	–	–	+3	-4
`PhysLean.lean`	1	–	–	+3	-1
`docs/_data`	1	–	–	+2	-2

scripts	1	–	–	+2	-2
PhysLean/Electromagnetism	1	–	–	+1	-1
.codespellignore	1	–	–	+1	-0

1.4 New declarations by kind

Kind	Count
lemma	72
def	26
structure	3
instance	3

2 Details by file

2.1 PhysLean.ClassicalMechanics.DampedHarmonicOscillator.Basic

PhysLean/ClassicalMechanics/DampedHarmonicOscillator/Basic.lean on GitHub.

structure DampedHarmonicOscillator [source]

Placeholder structure for the damped harmonic oscillator. The damped harmonic oscillator is specified by a mass `m`, a spring constant `k`, and a damping coefficient `γ`. All parameters are assumed to be positive (or non-negative for the damping coefficient).

```
structure DampedHarmonicOscillator where
  m : ℝ
  k : ℝ
  γ : ℝ
  m_pos : 0 < m
  k_pos : 0 < k
  γ_nonneg : 0 ≤ γ
```

lemma k_neq_zero [source]

```
@[simp]
lemma k_neq_zero : S.k ≠ 0
```

lemma m_neq_zero [source]

```
@[simp]
lemma m_neq_zero : S.m ≠ 0
```

def ω₀ [source]

The natural (undamped) angular frequency of the oscillator, $\omega_0 = \sqrt{(k/m)}$.

```
noncomputable def ω₀ : ℝ
```

lemma ω₀_pos [source]

```
@[simp]
lemma ω₀_pos : 0 < S.ω₀
```

lemma ω₀_sq [source]

```
lemma ω₀_sq : S.ω₀^2 = S.k / S.m
```

def EquationOfMotion [source]

The equation of motion for the damped harmonic oscillator.

A function `x : Time → ℝ` is a solution if it satisfies

$S.m * x¨ + S.\gamma * \dot{x} + S.k * x = 0$

for all times `t`.

```
noncomputable def EquationOfMotion (x : Time → ℝ) : Prop
```

def kineticEnergy [source]

The kinetic energy of the damped harmonic oscillator.

```
noncomputable def kineticEnergy (x : Time → ℝ) : Time → ℝ
```

def potentialEnergy [source]

The potential energy of the damped harmonic oscillator.

```
noncomputable def potentialEnergy (x : Time → ℝ) : Time → ℝ
```

def energy [source]

Mechanical energy of the damped harmonic oscillator.

```
noncomputable def energy (x : Time → ℝ) : Time → ℝ
```

def energyDissipationRate [source]

Energy dissipation rate along a trajectory `x : Time → ℝ`.

if `x` satisfies `S.equationOfMotion x`, then

*Time.deriv (S.energy x) t = - S.γ * (Time.deriv x t)^2,*

so the energy is non-increasing and not conserved when `S.γ > 0`.

```
noncomputable def energyDissipationRate (x : Time → ℝ) : Time → ℝ
```

def discriminant [source]

The discriminant that determines the damping regime.

```
noncomputable def discriminant : ℝ
```

def IsUnderdamped [source]

The system is underdamped when $\gamma^2 < 4mk$.

```
def IsUnderdamped : Prop
```

def IsCriticallyDamped [source]

The system is critically damped when $\gamma^2 = 4mk$.

```
def IsCriticallyDamped : Prop
```

def IsOverdamped [source]

The system is overdamped when $\gamma^2 > 4mk$.

```
def IsOverdamped : Prop
```

2.2 PhysLean.ClassicalMechanics.HarmonicOscillator.Solution

PhysLean/ClassicalMechanics/HarmonicOscillator/Solution.lean on GitHub.

structure InitialConditionsAtTime [source]

Initial conditions for the harmonic oscillator specified at an arbitrary time `t₀`.

This structure allows specifying the position and velocity at any time `t₀`, not necessarily at `t=0`. This is useful for problems where the natural reference time is not zero.

The conditions can be converted to the standard `InitialConditions` format (at `t=0`) using the `toInitialConditions` function.

```
structure InitialConditionsAtTime where
  t₀ : Time
  x_t₀ : EuclideanSpace ℝ (Fin 1)
  v_t₀ : EuclideanSpace ℝ (Fin 1)
```

lemma ext [source]

```
@[ext]
lemma InitialConditionsAtTime.ext {IC₁ IC₂ : InitialConditionsAtTime}
    (h1 : IC₁.t₀ = IC₂.t₀) (h2 : IC₁.x_t₀ = IC₂.x_t₀) (h3 : IC₁.v_t₀ = IC₂.v_t₀) :
    IC₁ = IC₂
```

def toInitialConditions [source]

Convert initial conditions at time `t₀` to standard initial conditions at `t=0`.

This conversion uses the harmonic oscillator solution formula with time-reversal. The resulting `InitialConditions` will produce a trajectory that passes through `x_t₀` with velocity `v_t₀` at time `t₀`.

See `toInitialConditions_trajectory_at_t₀` and `toInitialConditions_velocity_at_t₀` for the correctness proofs.

```
noncomputable def toInitialConditions (S : HarmonicOscillator)
    (IC : InitialConditionsAtTime) : InitialConditions where
```

lemma toInitialConditions_trajectory_at_t₀ [source]

The trajectory resulting from `toInitialConditions` passes through the specified position `x_t₀` at time `t₀`.

```
@[simp]
lemma toInitialConditions_trajectory_at_t₀ (S : HarmonicOscillator)
    (IC : InitialConditionsAtTime) :
    (IC.toInitialConditions S).trajectory S IC.t₀ = IC.x_t₀
```

lemma toInitialConditions_velocity_at_t₀ [source]

The trajectory resulting from `toInitialConditions` has the specified velocity `v_t₀` at time `t₀`.

```
@[simp]
lemma toInitialConditions_velocity_at_t₀ (S : HarmonicOscillator)
    (IC : InitialConditionsAtTime) :
    ∂ₜ ((IC.toInitialConditions S).trajectory S) IC.t₀ = IC.v_t₀
```

lemma toInitialConditions_energy_at_t₀ [source]

The energy of the trajectory at time `t₀` equals the energy computed from the initial conditions at `t₀`.

```
lemma toInitialConditions_energy_at_t₀ (S : HarmonicOscillator)
    (IC : InitialConditionsAtTime) :
    S.energy ((IC.toInitialConditions S).trajectory S) IC.t₀ =
    1/2 * (S.m * ‖IC.v_t₀‖^2 + S.k * ‖IC.x_t₀‖^2)
```

2.3 PhysLean.ClassicalMechanics.Lagrangian.TotalDerivativeEquivalence

PhysLean/ClassicalMechanics/Lagrangian/TotalDerivativeEquivalence.lean on GitHub.

def IsTotalTimeDerivative [source]

A function δL(r, v, t) is a total time derivative if it can be written as d/dt F(r, t) for some function F that depends on position and time but not velocity.

Mathematically: δL(r, v, t) = fderiv ℝ F (r, t) (v, 1)

By the chain rule, this equals ∂F/∂t(r, t) + ⟨∇ᵣF(r, t), v⟩.

This is the most general form of Lagrangian equivalence under total derivatives. The key point is that F must be independent of velocity.

```
def IsTotalTimeDerivative {n : ℕ}
    (δL : EuclideanSpace ℝ (Fin n) → EuclideanSpace ℝ (Fin n) → ℝ → ℝ) : Prop
```

lemma isTotalTimeDerivativeVelocity [source]

A velocity-only function that is a total time derivative must be linear in velocity.

If δL depends only on velocity and equals $d/dt\ F(r,\ t)$ for some F, then $\delta L(v) = \langle g,\ v\rangle$ for some constant vector g.

This characterization comes from the requirement that: - $d/dt\ F(r,\ t) = \partial F/\partial t + \langle \nabla F,\ \dot{r}\rangle = \partial F/\partial t + \langle \nabla F,\ v\rangle$ - For the result to be independent of r and t, we need $\nabla F = g$ (constant) and $\partial F/\partial t = 0$ - Thus $\delta L(v) = \langle g,\ v\rangle$

WLOG, we assume `δL 0 = 0` since constants are total derivatives ($c = d/dt(c{\cdot}t)$) and can be absorbed without affecting the equations of motion.

```
lemma isTotalTimeDerivativeVelocity {n : ℕ}
    (δL : EuclideanSpace ℝ (Fin n) → ℝ)
    (hδL0 : δL 0 = 0)
    (h : IsTotalTimeDerivative (fun _ v _ => δL v)) :
    ∃ g : EuclideanSpace ℝ (Fin n), ∀ v, δL v = ⟪g, v⟫_ℝ
```

2.4 PhysLean.CondensedMatter.TightBindingChain.Basic

PhysLean/CondensedMatter/TightBindingChain/Basic.lean on GitHub.

lemma localizedComp_adjoint [source]

The adjoint of localizedComp $|m\rangle\langle n|$ is $|n\rangle\langle m|$.

```
lemma localizedComp_adjoint (m n : Fin T.N) (ψ φ : T.HilbertSpace) :
    ⟪|m⟩⟨n| ψ, φ⟫_ℂ = ⟪ψ, |n⟩⟨m| φ⟫_ℂ := by
  simp only [localizedComp, LinearMap.coe_mk, AddHom.coe_mk]
  rw [inner_smul_left, inner_smul_right, inner_conj_symm]
  ring

noncomputable def hamiltonian : T.HilbertSpace →ₗ[ℂ] T.HilbertSpace :=
  T.E0 • ∑ n : Fin T.N, |n⟩⟨n| - T.t • ∑ n : Fin T.N, (|n⟩⟨n + 1| + |n + 1⟩⟨n|)

lemma hamiltonian_hermitian (ψ φ : T.HilbertSpace) :
    ⟪T.hamiltonian ψ, φ⟫_ℂ = ⟪ψ, T.hamiltonian φ⟫_ℂ := by
  simp only [hamiltonian, LinearMap.sub_apply, LinearMap.smul_apply, LinearMap.coe_sum,
    Finset.sum_apply, LinearMap.add_apply]
```

```
  rw [inner_sub_left, inner_sub_right]
  congr 1
  ·
    simp only [Finset.smul_sum]
    rw [sum_inner, inner_sum]
    apply Finset.sum_congr rfl
    intro n _
    simp only [inner_smul_left_eq_smul, inner_smul_right_eq_smul]
    rw [localizedComp_adjoint]
  ·
```

2.5 PhysLean.Particles.BeyondTheStandardModel.TwoHDM.Basic

PhysLean/Particles/BeyondTheStandardModel/TwoHDM/Basic.lean on GitHub.

lemma ext_of_fst_snd [source]

```
@[ext]
lemma ext_of_fst_snd {H1 H2 : TwoHiggsDoublet}
    (h1 : H1.Φ1 = H2.Φ1) (h2 : H1.Φ2 = H2.Φ2) : H1 = H2
```

instance SMul StandardModel.GaugeGroupI TwoHiggsDoublet [source]

```
noncomputable instance : SMul StandardModel.GaugeGroupI TwoHiggsDoublet where
```

lemma gaugeGroupI_smul_fst [source]

```
@[simp]
lemma gaugeGroupI_smul_fst (g : StandardModel.GaugeGroupI) (H : TwoHiggsDoublet) :
    (g • H).Φ1 = g • H.Φ1
```

lemma gaugeGroupI_smul_snd [source]

```
@[simp]
lemma gaugeGroupI_smul_snd (g : StandardModel.GaugeGroupI) (H : TwoHiggsDoublet) :
    (g • H).Φ2 = g • H.Φ2
```

instance MulAction StandardModel.GaugeGroupI TwoHiggsDoublet [source]

```
noncomputable instance : MulAction StandardModel.GaugeGroupI TwoHiggsDoublet where
```

2.6 PhysLean.Particles.BeyondTheStandardModel.TwoHDM.GramMatrix

PhysLean/Particles/BeyondTheStandardModel/TwoHDM/GramMatrix.lean on GitHub.

def gramMatrix [source]

The Gram matrix of the two Higgs doublet. This matrix is used in https://arxiv.org/abs/hep-ph/0605184.

```
noncomputable def gramMatrix (H : TwoHiggsDoublet) : Matrix (Fin 2) (Fin 2) ℂ
```

lemma gramMatrix_selfAdjoint [source]

```
lemma gramMatrix_selfAdjoint (H : TwoHiggsDoublet) :
    IsSelfAdjoint (gramMatrix H)
```

lemma eq_fst_norm_of_eq_gramMatrix [source]

```
lemma eq_fst_norm_of_eq_gramMatrix {H1 H2 : TwoHiggsDoublet}
    (h : H1.gramMatrix = H2.gramMatrix) : ‖H1.Φ1‖ = ‖H2.Φ1‖
```

lemma eq_snd_norm_of_eq_gramMatrix [source]

```
lemma eq_snd_norm_of_eq_gramMatrix {H1 H2 : TwoHiggsDoublet}
    (h : H1.gramMatrix = H2.gramMatrix) : ‖H1.Φ2‖ = ‖H2.Φ2‖
```

lemma gaugeGroupI_smul_gramMatrix [source]

```
@[simp]
lemma gaugeGroupI_smul_gramMatrix (g : StandardModel.GaugeGroupI) (H : TwoHiggsDoublet) :
    (g • H).gramMatrix = H.gramMatrix
```

lemma gramMatrix_det_eq [source]

```
lemma gramMatrix_det_eq (H : TwoHiggsDoublet) :
    H.gramMatrix.det = ‖H.Φ1‖ ^ 2 * ‖H.Φ2‖ ^ 2 - ‖⟪H.Φ1, H.Φ2⟫_ℂ‖ ^ 2
```

lemma gramMatrix_det_eq_real [source]

```
lemma gramMatrix_det_eq_real (H : TwoHiggsDoublet) :
    H.gramMatrix.det.re = ‖H.Φ1‖ ^ 2 * ‖H.Φ2‖ ^ 2 - ‖⟪H.Φ1, H.Φ2⟫_ℂ‖ ^ 2
```

lemma gramMatrix_det_nonneg [source]

```
lemma gramMatrix_det_nonneg (H : TwoHiggsDoublet) :
    0 ≤ H.gramMatrix.det.re
```

lemma gramMatrix_tr_nonneg [source]

```
lemma gramMatrix_tr_nonneg (H : TwoHiggsDoublet) :
    0 ≤ H.gramMatrix.trace.re
```

lemma gaugeGroupI_exists_fst_eq [source]

```
lemma gaugeGroupI_exists_fst_eq {H : TwoHiggsDoublet} (h1 : H.Φ1 ≠ 0) :
    ∃ g : StandardModel.GaugeGroupI,
      g • H.Φ1 = (!₂[‖H.Φ1‖, 0] : HiggsVec) ∧
      (g • H.Φ2) 0 = ⟪H.Φ1, H.Φ2⟫_ℂ / ‖H.Φ1‖ ∧
      ‖(g • H.Φ2) 1‖ = Real.sqrt (H.gramMatrix.det.re) / ‖H.Φ1‖
```

lemma gaugeGroupI_exists_fst_eq_snd_eq [source]

```
lemma gaugeGroupI_exists_fst_eq_snd_eq {H : TwoHiggsDoublet} (h1 : H.Φ1 ≠ 0) :
    ∃ g : StandardModel.GaugeGroupI,
      g • H.Φ1 = (!₂[‖H.Φ1‖, 0] : HiggsVec) ∧
      g • H.Φ2 = (!₂[⟪H.Φ1, H.Φ2⟫_ℂ / ‖H.Φ1‖, √(H.gramMatrix.det.re) / ‖H.Φ1‖] : HiggsVec)
```

lemma mem_orbit_gaugeGroupI_iff_gramMatrix [source]

```
lemma mem_orbit_gaugeGroupI_iff_gramMatrix (H1 H2 : TwoHiggsDoublet) :
    H1 ∈ MulAction.orbit GaugeGroupI H2 ↔ H1.gramMatrix = H2.gramMatrix
```

lemma gramMatrix_surjective_det_tr [source]

```
lemma gramMatrix_surjective_det_tr (K : Matrix (Fin 2) (Fin 2) ℂ)
    (hKs : IsSelfAdjoint K) (hKdet : 0 ≤ K.det.re) (hKtr : 0 ≤ K.trace.re) :
    ∃ H : TwoHiggsDoublet, H.gramMatrix = K
```

def gramVector [source]

A real vector containing the components of the Gram matrix in the Pauli basis.

```
noncomputable def gramVector (H : TwoHiggsDoublet) : Fin 1 ⊕ Fin 3 → ℝ
```

lemma gaugeGroupI_smul_fst_gramVector [source]

```
@[simp]
lemma gaugeGroupI_smul_fst_gramVector (g : StandardModel.GaugeGroupI)
    (H : TwoHiggsDoublet) (μ : Fin 1 ⊕ Fin 3) :
    (g • H).gramVector μ = H.gramVector μ
```

lemma gramMatrix_eq_gramVector_sum_pauliMatrix [source]

```
lemma gramMatrix_eq_gramVector_sum_pauliMatrix (H : TwoHiggsDoublet) :
    gramMatrix H = (1 / 2 : ℝ) • ∑ μ, H.gramVector μ • PauliMatrix.pauliMatrix μ
```

lemma gramMatrix_eq_component_gramVector [source]

```
lemma gramMatrix_eq_component_gramVector (H : TwoHiggsDoublet) :
    gramMatrix H =
    !![(1 / 2 : ℂ) * (H.gramVector (Sum.inl 0) + H.gramVector (Sum.inr 2)),
      (1 / 2 : ℂ) * (H.gramVector (Sum.inr 0) - Complex.I * H.gramVector (Sum.inr 1));
      (1 / 2 : ℂ) * (H.gramVector (Sum.inr 0) + Complex.I * H.gramVector (Sum.inr 1)),
      (1 / 2 : ℂ) * (H.gramVector (Sum.inl 0) - H.gramVector (Sum.inr 2))]
```

lemma gramVector_inl_eq_trace_gramMatrix [source]

```
lemma gramVector_inl_eq_trace_gramMatrix (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inl 0) = H.gramMatrix.trace.re
```

lemma gramVector_inl_nonneg [source]

```
lemma gramVector_inl_nonneg (H : TwoHiggsDoublet) :
    0 ≤ H.gramVector (Sum.inl 0)
```

lemma normSq_Φ1_eq_gramVector [source]

```
lemma normSq_Φ1_eq_gramVector (H : TwoHiggsDoublet) :
    ‖H.Φ1‖ ^ 2 = (1/2 : ℝ) * (H.gramVector (Sum.inl 0) + H.gramVector (Sum.inr 2))
```

lemma normSq_Φ2_eq_gramVector [source]

```
lemma normSq_Φ2_eq_gramVector (H : TwoHiggsDoublet) :
    ‖H.Φ2‖ ^ 2 = (1/2 : ℝ) * (H.gramVector (Sum.inl 0) - H.gramVector (Sum.inr 2))
```

lemma Φ1_inner_Φ2_eq_gramVector [source]

```
lemma Φ1_inner_Φ2_eq_gramVector (H : TwoHiggsDoublet) :
    (⟪H.Φ1, H.Φ2⟫_ℂ) = (1/2 : ℝ) * (H.gramVector (Sum.inr 0) +
    Complex.I * H.gramVector (Sum.inr 1))
```

lemma Φ2_inner_Φ1_eq_gramVector [source]

```
lemma Φ2_inner_Φ1_eq_gramVector (H : TwoHiggsDoublet) :
    (⟪H.Φ2, H.Φ1⟫_ℂ) = (1/2 : ℝ) * (H.gramVector (Sum.inr 0) -
    Complex.I * H.gramVector (Sum.inr 1))
```

lemma Φ1_inner_Φ2_normSq_eq_gramVector [source]

```
lemma Φ1_inner_Φ2_normSq_eq_gramVector (H : TwoHiggsDoublet) :
    ‖⟪H.Φ1, H.Φ2⟫_ℂ‖ ^ 2 =
    (1/4 : ℝ) * (H.gramVector (Sum.inr 0) ^ 2 + H.gramVector (Sum.inr 1) ^ 2)
```

lemma gramVector_inl_zero_eq [source]

```
lemma gramVector_inl_zero_eq (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inl 0) = ‖H.Φ1‖ ^ 2 + ‖H.Φ2‖ ^ 2
```

lemma gramVector_inl_zero_eq_gramMatrix [source]

```
lemma gramVector_inl_zero_eq_gramMatrix (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inl 0) = (H.gramMatrix 0 0).re + (H.gramMatrix 1 1).re
```

lemma gramVector_inr_zero_eq [source]

```
lemma gramVector_inr_zero_eq (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inr 0) = 2 * (⟪H.Φ1, H.Φ2⟫_ℂ).re
```

lemma gramVector_inr_zero_eq_gramMatrix [source]

```
lemma gramVector_inr_zero_eq_gramMatrix (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inr 0) = 2 * (H.gramMatrix 1 0).re
```

lemma gramVector_inr_one_eq [source]

```
lemma gramVector_inr_one_eq (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inr 1) = 2 * (⟪H.Φ1, H.Φ2⟫_ℂ).im
```

lemma gramVector_inr_one_eq_gramMatrix [source]

```
lemma gramVector_inr_one_eq_gramMatrix (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inr 1) = 2 * (H.gramMatrix 1 0).im
```

lemma gramVector_inr_two_eq [source]

```
lemma gramVector_inr_two_eq (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inr 2) = ‖H.Φ1‖ ^ 2 - ‖H.Φ2‖ ^ 2
```

lemma gramVector_inr_two_eq_gramMatrix [source]

```
lemma gramVector_inr_two_eq_gramMatrix (H : TwoHiggsDoublet) :
    H.gramVector (Sum.inr 2) = (H.gramMatrix 0 0).re - (H.gramMatrix 1 1).re
```

lemma gramMatrix_det_eq_gramVector [source]

```
lemma gramMatrix_det_eq_gramVector (H : TwoHiggsDoublet) :
    H.gramMatrix.det.re =
    (1/4 : ℝ) * (H.gramVector (Sum.inl 0) ^ 2 -
    ∑ μ : Fin 3, H.gramVector (Sum.inr μ) ^ 2)
```

lemma gramVector_inr_sum_sq_le_inl [source]

```
lemma gramVector_inr_sum_sq_le_inl (H : TwoHiggsDoublet) :
    ∑ μ : Fin 3, H.gramVector (Sum.inr μ) ^ 2 ≤ H.gramVector (Sum.inl 0) ^ 2
```

lemma gramVector_surjective [source]

```
lemma gramVector_surjective (v : Fin 1 ⊕ Fin 3 → ℝ)
    (h_inl : 0 ≤ v (Sum.inl 0))
    (h_det : ∑ μ : Fin 3, v (Sum.inr μ) ^ 2 ≤ v (Sum.inl 0) ^ 2) :
    ∃ H : TwoHiggsDoublet, H.gramVector = v
```

2.7 PhysLean.Particles.BeyondTheStandardModel.TwoHDM.Potential

PhysLean/Particles/BeyondTheStandardModel/TwoHDM/Potential.lean on GitHub.

structure PotentialParameters [source]

The parameters of the Two Higgs doublet model potential. Following the convention of https://arxiv.org/pdf/1605.03237.

```
structure PotentialParameters where
  m₁₁2 : ℝ
  m₂₂2 : ℝ
  m₁₂2 : ℂ
  ↕₁ : ℝ
```

```
↕₂ : ℝ
↕₃ : ℝ
↕₄ : ℝ
↕₅ : ℂ
↕₆ : ℂ
↕₇ : ℂ
```

def ξ [source]

A reparameterization of the parameters of the quadratic terms of the potential for use with the gramVector.

```
noncomputable def ξ (P : PotentialParameters) : Fin 1 ⊕ Fin 3 → ℝ
```

def η [source]

A reparameterization of the parameters of the quartic terms of the potential for use with the gramVector.

```
noncomputable def η (P : PotentialParameters) : Fin 1 ⊕ Fin 3 → Fin 1 ⊕ Fin 3 → ℝ
  | Sum.inl 0, Sum.inl 0 => (P.↕₁ + P.↕₂ + 2 * P.↕₃) / 8
  | Sum.inl 0, Sum.inr 0 => (P.↕₆.re + P.↕₇.re) * (1 / 4)
  | Sum.inl 0, Sum.inr 1 => (P.↕₆.im + P.↕₇.im) * (-1 / 4)
  | Sum.inl 0, Sum.inr 2 => (P.↕₁ - P.↕₂) * (1 / 8)
  | Sum.inr 0, Sum.inl 0 => (P.↕₆.re + P.↕₇.re) * (1 / 4)
  | Sum.inr 1, Sum.inl 0 => (P.↕₆.im + P.↕₇.im) * (-1 / 4)
  | Sum.inr 2, Sum.inl 0 => (P.↕₁ - P.↕₂) * (1 / 8)
```

lemma η_symm [source]

```
lemma η_symm (P : PotentialParameters) (μ ν : Fin 1 ⊕ Fin 3) :
    P.η μ ν = P.η ν μ
```

def massTerm [source]

The mass term of the two Higgs doublet model potential.

```
noncomputable def massTerm (P : PotentialParameters) (H : TwoHiggsDoublet) : ℝ
```

lemma massTerm_eq_gramVector [source]

```
lemma massTerm_eq_gramVector (P : PotentialParameters) (H : TwoHiggsDoublet) :
    massTerm P H = ∑ μ, P.ξ μ * H.gramVector μ
```

lemma gaugeGroupI_smul_massTerm [source]

```
@[simp]
lemma gaugeGroupI_smul_massTerm (g : StandardModel.GaugeGroupI) (P : PotentialParameters)
    (H : TwoHiggsDoublet) :
    massTerm P (g • H) = massTerm P H
```

def quarticTerm [source]

The quartic term of the two Higgs doublet model potential.

```
noncomputable def quarticTerm (P : PotentialParameters) (H : TwoHiggsDoublet) : ℝ
```

lemma quarticTerm_↕₄_expand [source]

```
lemma quarticTerm_↕₄_expand (P : PotentialParameters) (H : TwoHiggsDoublet) :
    H.quarticTerm P =
    1/2 * P.↕₁ * ‖H.Φ1‖ ^ 2 * ‖H.Φ1‖ ^ 2 + 1/2 * P.↕₂ * ‖H.Φ2‖ ^ 2 * ‖H.Φ2‖ ^ 2
    + P.↕₃ * ‖H.Φ1‖ ^ 2 * ‖H.Φ2‖ ^ 2
    + P.↕₄ * (⟪H.Φ1, H.Φ2⟫_ℂ * ⟪H.Φ2, H.Φ1⟫_ℂ).re
    + (1/2 * P.↕₅ * ⟪H.Φ1, H.Φ2⟫_ℂ ^ 2 + 1/2 * conj P.↕₅ * ⟪H.Φ2, H.Φ1⟫_ℂ ^ 2).re
    + (P.↕₆ * ‖H.Φ1‖ ^ 2 * ⟪H.Φ1, H.Φ2⟫_ℂ + conj P.↕₆ * ‖H.Φ1‖ ^ 2 * ⟪H.Φ2, H.Φ1⟫_ℂ).re
    + (P.↕₇ * ‖H.Φ2‖ ^ 2 * ⟪H.Φ1, H.Φ2⟫_ℂ + conj P.↕₇ * ‖H.Φ2‖ ^ 2 * ⟪H.Φ2, H.Φ1⟫_ℂ).re
```

lemma quarticTerm_eq_gramVector [source]

```
lemma quarticTerm_eq_gramVector (P : PotentialParameters) (H : TwoHiggsDoublet) :
    quarticTerm P H = ∑ a, ∑ b, H.gramVector a * H.gramVector b * P.η a b
```

lemma gaugeGroupI_smul_quarticTerm [source]

```
@[simp]
lemma gaugeGroupI_smul_quarticTerm (g : StandardModel.GaugeGroupI) (P : PotentialParameters)
    (H : TwoHiggsDoublet) :
    quarticTerm P (g • H) = quarticTerm P H
```

def potential [source]

The potential of the two Higgs doublet model.

```
noncomputable def potential (P : PotentialParameters) (H : TwoHiggsDoublet) : ℝ
```

lemma gaugeGroupI_smul_potential [source]

```
@[simp]
lemma gaugeGroupI_smul_potential (g : StandardModel.GaugeGroupI)
    (P : PotentialParameters) (H : TwoHiggsDoublet) :
    potential P (g • H) = potential P H
```

def PotentialIsBounded [source]

The condition that the potential is bounded from below.

```
def PotentialIsBounded (P : PotentialParameters) : Prop
```

lemma potentialIsBounded_iff_forall_gramVector [source]

```
lemma potentialIsBounded_iff_forall_gramVector (P : PotentialParameters) :
    PotentialIsBounded P ↔ ∃ c : ℝ, ∀ K : Fin 1 ⊕ Fin 3 → ℝ, 0 ≤ K (Sum.inl 0) →
      ∑ μ : Fin 3, K (Sum.inr μ) ^ 2 ≤ K (Sum.inl 0) ^ 2 →
      c ≤ ∑ μ, P.ξ μ * K μ + ∑ a, ∑ b, K a * K b * P.η a b
```

lemma potentialIsBounded_iff_forall_euclid [source]

```
lemma potentialIsBounded_iff_forall_euclid (P : PotentialParameters) :
    PotentialIsBounded P ↔ ∃ c, ∀ K0 : ℝ, ∀ K : EuclideanSpace ℝ (Fin 3), 0 ≤ K0 →
      ‖K‖ ^ 2 ≤ K0 ^ 2 → c ≤ P.ξ (Sum.inl 0) * K0 + ∑ μ, P.ξ (Sum.inr μ) * K μ
      + K0 ^ 2 * P.η (Sum.inl 0) (Sum.inl 0)
      + 2 * K0 * ∑ b, K b * P.η (Sum.inl 0) (Sum.inr b) +
      ∑ a, ∑ b, K a * K b * P.η (Sum.inr a) (Sum.inr b)
```

lemma potentialIsBounded_iff_forall_euclid_lt [source]

```
lemma potentialIsBounded_iff_forall_euclid_lt (P : PotentialParameters) :
    PotentialIsBounded P ↔ ∃ c ≤ 0, ∀ K0 : ℝ, ∀ K : EuclideanSpace ℝ (Fin 3), 0 < K0 →
      ‖K‖ ^ 2 ≤ K0 ^ 2 → c ≤ P.ξ (Sum.inl 0) * K0 + ∑ μ, P.ξ (Sum.inr μ) * K μ
      + K0 ^ 2 * P.η (Sum.inl 0) (Sum.inl 0)
      + 2 * K0 * ∑ b, K b * P.η (Sum.inl 0) (Sum.inr b) +
      ∑ a, ∑ b, K a * K b * P.η (Sum.inr a) (Sum.inr b)
```

def massTermReduced [source]

A function related to the mass term of the potential, used in the boundedness condition and equivalent to the term `J2` in https://arxiv.org/abs/hep-ph/0605184.

```
noncomputable def massTermReduced (P : PotentialParameters) (k : EuclideanSpace ℝ (Fin 3)) : ℝ
```

lemma massTermReduced_lower_bound [source]

```
lemma massTermReduced_lower_bound (P : PotentialParameters) (k : EuclideanSpace ℝ (Fin 3))
    (hk : ‖k‖ ^ 2 ≤ 1) : P.ξ (Sum.inl 0) - √(∑ a, |P.ξ (Sum.inr a)| ^ 2) ≤ massTermReduced P k
```

def quarticTermReduced [source]

A function related to the quartic term of the potential, used in the boundedness condition and equivalent to the term `J4` in https://arxiv.org/abs/hep-ph/0605184.

```
noncomputable def quarticTermReduced (P : PotentialParameters) (k : EuclideanSpace ℝ (Fin 3)) :
    ℝ
```

lemma potentialIsBounded_iff_exists_forall_forall_reduced [source]

```
lemma potentialIsBounded_iff_exists_forall_forall_reduced (P : PotentialParameters) :
    PotentialIsBounded P ↔ ∃ c ≤ 0, ∀ K0 : ℝ, ∀ k : EuclideanSpace ℝ (Fin 3), 0 < K0 →
      ‖k‖ ^ 2 ≤ 1 → c ≤ K0 * massTermReduced P k + K0 ^ 2 * quarticTermReduced P k
```

lemma quarticTermReduced_nonneg_of_potentialIsBounded [source]

```
lemma quarticTermReduced_nonneg_of_potentialIsBounded (P : PotentialParameters)
    (hP : PotentialIsBounded P) (k : EuclideanSpace ℝ (Fin 3))
    (hk : ‖k‖ ^ 2 ≤ 1) : 0 ≤ quarticTermReduced P k
```

lemma potentialIsBounded_iff_exists_forall_reduced [source]

```
lemma potentialIsBounded_iff_exists_forall_reduced (P : PotentialParameters) :
    PotentialIsBounded P ↔ ∃ c, 0 ≤ c ∧ ∀ k : EuclideanSpace ℝ (Fin 3), ‖k‖ ^ 2 ≤ 1 →
      0 ≤ quarticTermReduced P k ∧
      (massTermReduced P k < 0 →
      massTermReduced P k ^ 2 ≤ 4 * quarticTermReduced P k * c)
```

lemma massTermReduced_pos_of_quarticTermReduced_zero_potentialIsBounded [source]

```
lemma massTermReduced_pos_of_quarticTermReduced_zero_potentialIsBounded (P :
    PotentialParameters)
    (hP : PotentialIsBounded P) (k : EuclideanSpace ℝ (Fin 3))
    (hk : ‖k‖ ^ 2 ≤ 1) (hq : quarticTermReduced P k = 0) : 0 ≤ massTermReduced P k
```

lemma potentialIsBounded_iff_forall_reduced [source]

```
@[sorryful]
lemma potentialIsBounded_iff_forall_reduced (P : PotentialParameters) :
    PotentialIsBounded P ↔ ∀ k : EuclideanSpace ℝ (Fin 3), ‖k‖ ^ 2 ≤ 1 →
      0 ≤ quarticTermReduced P k ∧ (quarticTermReduced P k = 0 → 0 ≤ massTermReduced P k)
```

2.8 PhysLean.Particles.StandardModel.Basic

PhysLean/Particles/StandardModel/Basic.lean on GitHub.

def ofU1Subgroup [source]

The inclusion of a U(1) subgroup.

```
def ofU1Subgroup (u1 : unitary ℂ) : GaugeGroupI
```

lemma ofU1Subgroup_toSU3 [source]

```
@[simp]
lemma ofU1Subgroup_toSU3 (u1 : unitary ℂ) :
    toSU3 (ofU1Subgroup u1) = 1
```

lemma ofU1Subgroup_toSU2 [source]

```
@[simp]
lemma ofU1Subgroup_toSU2 (u1 : unitary ℂ) :
    toSU2 (ofU1Subgroup u1) = (!![star (u1 ^ 3 : unitary ℂ), 0;0, (u1 ^ 3 : unitary ℂ)], by
    simp only [SetLike.mem_coe]
    rw [mem_unitaryGroup_iff']
    funext i j
    rw [Matrix.mul_apply]
    fin_cases i <;> fin_cases j <;> simp [conj_mul'], by
    simp only [RCLike.star_def, SetLike.mem_coe, MonoidHom.mem_mker, coe_detMonoidHom,
      det_fin_two_of, conj_mul', mul_zero, sub_zero]
    simp)
```

lemma ofU1Subgroup_toU1 [source]

```
@[simp]
lemma ofU1Subgroup_toU1 (u1 : unitary ℂ) :
    toU1 (ofU1Subgroup u1) = u1
```

2.9 PhysLean.Particles.StandardModel.HiggsBoson.Basic

PhysLean/Particles/StandardModel/HiggsBoson/Basic.lean on GitHub.

lemma gaugeGroupI_smul_eq_U1_smul_SU2 [source]

```
lemma gaugeGroupI_smul_eq_U1_smul_SU2 (g : StandardModel.GaugeGroupI) (φ : HiggsVec) :
    g • φ = (WithLp.toLp 2 <| (g.toU1 ^ 3 • g.toSU2.1) *ᵥ φ.ofLp)
```

instance DistribMulAction StandardModel.GaugeGroupI HiggsVec [source]

```
instance : DistribMulAction StandardModel.GaugeGroupI HiggsVec where
```

lemma ofU1Subgroup_smul_eq_smul [source]

```
lemma ofU1Subgroup_smul_eq_smul (g : unitary ℂ) (φ : HiggsVec) :
    (StandardModel.GaugeGroupI.ofU1Subgroup g) • φ =
    (WithLp.toLp 2 <| !![1, 0; 0, g.1 ^ 6] *ᵥ φ.ofLp)
```

lemma gaugeGroupI_smul_phase_snd [source]

```
lemma gaugeGroupI_smul_phase_snd (φ : HiggsVec) :
    ∃ g : StandardModel.GaugeGroupI,
      (g • φ).ofLp 1 = ‖(φ.ofLp 1)‖ ∧
      (∀ φ1 : HiggsVec, (g • φ1).ofLp 0 = φ1.ofLp 0) ∧
      (∀ a : ℝ, g • (!₂[a, 0] : HiggsVec) = (!₂[a, 0] : HiggsVec))
```

2.10 PhysLean.Relativity.Tensors.RealTensor.ToComplex

PhysLean/Relativity/Tensors/RealTensor/ToComplex.lean on GitHub.

lemma colorToComplex_append [source]

`colorToComplex` commutes with `Fin.append` (as functions).

```
@[simp]
lemma colorToComplex_append {n m : ℕ}
    (c : Fin n → realLorentzTensor.Color) (c1 : Fin m → realLorentzTensor.Color) :
    (colorToComplex ∘ Fin.append c c1) = Fin.append (colorToComplex ∘ c) (colorToComplex ∘ c1)
```

def prodTColorToComplex [source]

`prodT` on the complex side, with colors written as `colorToComplex ∘ Fin.append ...`. This is `prodT` followed by a cast using `colorToComplex_append`.

```
noncomputable def prodTColorToComplex {n m : ℕ}
    {c : Fin n → realLorentzTensor.Color} {c1 : Fin m → realLorentzTensor.Color} :
    ℂT(colorToComplex ∘ c) → ℂT(colorToComplex ∘ c1) → ℂT(colorToComplex ∘ Fin.append c c1)
```

lemma tau_colorToComplex [source]

τ commutes with `colorToComplex` on the Lorentz `up/down` colors.

```
@[simp]
lemma tau_colorToComplex (x : realLorentzTensor.Color) :
    (complexLorentzTensor).τ (colorToComplex x) = colorToComplex ((realLorentzTensor).τ x)
```

lemma complex_repDim_up [source]

```
@[simp]
lemma complex_repDim_up :
    (complexLorentzTensor).repDim complexLorentzTensor.Color.up = 4
```

lemma complex_repDim_down [source]

```
@[simp]
lemma complex_repDim_down :
    (complexLorentzTensor).repDim complexLorentzTensor.Color.down = 4
```

def evalIdxToComplex [source]

Convert an evaluation index from the real repDim to the complex repDim.

```
def evalIdxToComplex {n : ℕ}
    {c : Fin (n + 1) → realLorentzTensor.Color} (i : Fin (n + 1))
    (b : Fin ((realLorentzTensor).repDim (c i))) :
    Fin ((complexLorentzTensor).repDim ((colorToComplex ∘ c) i))
```

def evalTColorToComplex [source]

`evalT` on the complex side, but with output colors as `colorToComplex ∘ (c ∘ i.succAbove)`. Implemented via `permT (σ := id) (by simp)` as a transport.

```
noncomputable def evalTColorToComplex {n : ℕ}
    {c : Fin (n + 1) → realLorentzTensor.Color} (i : Fin (n + 1))
    (b : Fin ((realLorentzTensor).repDim (c i))) :
    ℂT(colorToComplex ∘ c) → ℂT(colorToComplex ∘ (c ∘ i.succAbove))
```