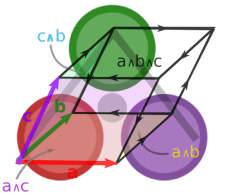


Differential geometric algebra: compute using Grassmann.jl, Cartan.jl

by Michael Reed 2019-2021-2025, Crucial Flow Research, *crucialflow.com*, *github.com/chakravala*, *Grassmann.jl* `v0.8.33`, *Cartan.jl* `v0.4.2`

Abstract (Dream Scatter)

Initially, *Grassmann.jl* and *Cartan.jl* introduced the *DirectSum* formalism into computational language design for differential geometric algebra, thus enabling the construction of custom algebras through metaprogramming. *Grassmann.jl* pioneered a novel type system design along with its syntax and semantics, which has undergone significant refinement through many years of development and continuous improvement. *Cartan.jl* represents a groundbreaking extension of capabilities offered by *Grassmann.jl*, marking a pioneering fully realized implementation of numerical differential geometric algebra, based on `TensorField` representations over a `FrameBundle` and the `ImmersedTopology` of the `FiberBundle`. *Grassmann.jl* and *Cartan.jl* build on Julia's multiple dispatch and metaprogramming capabilities, presenting a new computational language design approach to interfacing with differential geometric algebra based on a new *sector integral theorem*. This pioneering design not only actualizes but also elevates computational language syntax to new heights using the foundations of *Grassmann.jl* and *Cartan.jl*. The *Grassmann.jl* and *Cartan.jl* packages introduce pioneering computational language designs, having inspired imitation projects and thereby validating the project's relevance as significant advance in computational mathematics.

Packages *Grassmann.jl* and *Cartan.jl* can be used as universal language for finite element methods based on a discrete manifold bundle. Tools built on these foundations enable computations based on multi-linear algebra and spin groups using the geometric algebra known as Grassmann algebra or Clifford algebra. This foundation is built on a `DirectSum` parametric type system for tangent bundles and vector spaces generating the algorithms for local tangent algebras in a global context. Geometric algebra mathematical foundations for differential geometry can be used to simplify the Maxwell equations to a single wave equation due to the geometric product. With this unifying mathematical foundation, it is possible to improve efficiency of multi-disciplinary research using geometric tensor calculus by relying on universal mathematical principles. Tools built on this differential geometric algebra provide an excellent language for the newly presented *sector integral theorem*, the Helmholtz decomposition, and Hodge-DeRahm co/homology.

The *Grassmann.jl* package provides tools for computations based on multi-linear algebra and spin groups using the extended geometric algebra known as Grassmann-Clifford-Hodge algebra. Algebra operations include exterior, regressive, inner, and geometric, along with the Hodge star and boundary operators. Code generation enables concise usage of the algebra syntax. *DirectSum.jl* multivector parametric type polymorphism is based on tangent vector spaces and conformal projective geometry. Additionally, the universal interoperability between different sub-algebras is enabled by *AbstractTensors.jl*, on which the type system is built. The design is based on `TensorAlgebra{V}` abstract type interoperability from *AbstractTensors.jl* with a $\mathbb{K}$-module type parameter V from *DirectSum.jl*. Abstract vector space type operations happen at compile-time, resulting in a differential geometric algebra of multivector forms.

Building on the *Grassmann.jl* foundation, the *Cartan.jl* extension then defines `TensorField{B,F,N} <: GlobalFiber{LocalTensor{B,F},N}` for both a local `ProductSpace` and general `ImmersedTopology` specifications on any `FrameBundle` expressed with *Grassmann.jl* algebra. Many of these modular methods can work on input meshes or product topologies of any dimension, although there are some methods which are specialized. `Cartan` provides an algebra for `FiberBundle` sections and any associated bundles on a manifold in terms of `Grassmann` elements. Calculus of `Variation` fields can also be generated with the combined topology of a `FiberProductBundle`. Furthermore, the `FiberProduct` enables construction of `HomotopyBundle` types. The `Cartan` package standardizes composition of various methods and functors applied to specialized categories transformed in terms of a unified representation over a product topology, especially having fibers of the `Grassmann` algebra and using `Cartan` methods over a `FrameBundle`.

0.1. Direct sum parametric type polymorphism

Definition 0.1 (Vector space $\Lambda^1 V = V$ is a field's $\mathbb{K}$-module)**.** Let V be a $\mathbb{K}$-module (abelian group with respect to $+$) with an element $1 \in \mathbb{K}$ such that $1V = V$ by scalar multiplication $\mathbb{K} \times V \to V$ over field $\mathbb{K}$ satisfying

- $a(x + y) = ax + ay$ distribution of vector addition,
- $(a + b)x = ax + bd$ distribution of field addition,
- $(ab)x = a(bx)$ associative compatibility.

In the software package `Grassmann`, a generating vector space $\mathbb{K}$-module is specified as a value of `<:TensorBundle` (an abstract type).

The `AbstractTensors` package is intended for universal interoperation of the abstract `TensorAlgebra` type system. All `TensorAlgebra{V}` subtypes have type parameter V, used to store a `Submanifold{M}` value, which is parametrized by M the `TensorBundle` choice. This means that different tensor types can have a commonly shared underlying $\mathbb{K}$-module parametric type expressed by defining `V::Submanifold{M}`. Each `TensorAlgebra` subtype must be accompanied by a corresponding `TensorBundle` parameter, which is fully static at compile time. Due to the parametric type system for the $\mathbb{K}$-module types, the compiler can fully pre-allocate and often cache.

Additionally, a universal unit volume element can be specified in terms of `LinearAlgebra.UniformScaling`, which is independent of V and has its interpretation only instantiated by context of `TensorAlgebra{V}` elements being operated on. Interoperability of `LinearAlgebra.UniformScaling` as a pseudoscalar element which takes on the `TensorBundle` form of any other `TensorAlgebra` element is handled globally. This enables the usage of `I` from `LinearAlgebra` as a universal pseudoscalar element defined at every point x of a `Manifold`, which is mathematically denoted by $I = I(x)$ and specified by the $g(x)$ bilinear tensor field of TM. Utility methods such as `scalar`, `involute`, `norm`, `norm2`, `unit`, `even`, `odd` are also defined.

Definition 0.2 (Linear dependence)**.** Let V be a vector space over field $\mathbb{K}$, then the set $\{v_i\}_i$ is linearly dependent iff $\sum_{i=1}^{n} k_i v_i = 0$ for some $0 \neq k \in \mathbb{K}^n$.

Definition 0.3 ($\wedge$-product annihilation)**.** For linearly dependent $\{v_i\}_i^n \subset V$

$$v_1 \wedge v_2 \wedge \cdots \wedge v_n = 0$$

Initially, it is enough to understand that $\wedge : \Lambda^n V \times \Lambda^m V \to \Lambda^{n+m} V$ is an operation which is zero for linearly dependent arguments. However, this idea comes from extending Grassmann's product $v_i \wedge v_j = -v_j \wedge v_i \implies v_i \wedge v_i = 0 = -v_i \wedge v_i$ to yield a tool for characterizing linear dependence.

Definition 0.4 (Dimension n-Submanifold in $\Lambda^n V$)**.** Note that writing the product $v_1 \wedge v_2 \wedge \cdots \wedge v_n \neq 0$ implies a linearly independent set $\{v_i\}_1^n \subseteq V$ isomorphic to an n-`Submanifold`. Furthermore, $\mathbb{K} \times \{v_1 \wedge v_2 \wedge \cdots \wedge v_n\} \cong \mathbb{K}$ shows the 1-dimensional basis subspace is induced by any n-`Submanifold`.

Example 0.5. $\mathbb{K} = \Lambda^0\mathbb{K} \cong \Lambda^1\mathbb{K}$ is a vector space or 0-`Submanifold`.

Example 0.6. $\Lambda^n V$ is vector space with $\Lambda^1\Lambda^n V = \Lambda^n V$ and $\Lambda^0\Lambda^n V = \Lambda^0 V$.

Denote $V^* = V \setminus \{0\}$ as the set V excluding the 0 element in next:

Definition 0.7 (Direct sum $\oplus$)**.** To consider a set of linearly independent spaces, let $\pi_i : V \to V_i$ be projections with vector space $V_i \subset V_i$, define

$$V_1 \oplus V_2 \oplus \cdots \oplus V_n = V \iff \bigwedge : V_1^* \times V_2^* \times \cdots \times V_n^* \to \Lambda^n V^*.$$

Direct sum of a full non-zero product implies an n-`Submanifold`.

Definition 0.8. Grade m-projection `grade(x,m)` is $\langle \Lambda V \rangle_m = \Lambda^m V$ so

$$\Lambda V = \bigoplus_{m=0}^{n} \langle \Lambda V \rangle_m = \Lambda^0 V \oplus \Lambda^1 V \oplus \cdots \oplus \Lambda^n V, \qquad \langle \Lambda V \rangle_m = \bigoplus_{m=1}^{\binom{n}{m}} \mathbb{K}.$$

Note that $\dim \langle \Lambda V \rangle_m = \binom{n}{m}$ and hence $\dim \Lambda V = \sum_{m=0}^{n} \binom{n}{m} = 2^n$.

Example 0.9 (Combinatorics of power set $\mathcal{P}(V)$)**.** Let $v_1, v_2, v_3 \in \mathbb{R}^3$, then the power set of basis elements is:

$$\mathcal{P}(\mathbb{R}^3) = \{\emptyset, \{v_1\}, \{v_2\}, \{v_3\}, \{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_1, v_2, v_3\}\}$$

Form a direct sum over the elements of $\mathcal{P}(V)$ with $\wedge$ to define ΛV, e.g.

$$\Lambda(\mathbb{R}^3) = \Lambda^0(\mathbb{R}^3) \oplus \Lambda^1(\mathbb{R}^3) \oplus \Lambda^2(\mathbb{R}^3) \oplus \Lambda^3(\mathbb{R}^3)$$

$$\overbrace{v_\emptyset}^{\Lambda^0\mathbb{R}} \oplus \overbrace{v_1 \oplus v_2 \oplus v_3}^{\Lambda^1(\mathbb{R}^3)} \oplus \overbrace{(v_1 \wedge v_2) \oplus (v_1 \wedge v_3) \oplus (v_2 \wedge v_3)}^{\Lambda^2(\mathbb{R}^3)} \oplus \overbrace{(v_1 \wedge v_2 \wedge v_3)}^{\Lambda^3(\mathbb{R}^3)}$$

Definition 0.10. Let $M = T^\mu V$ be a $\mathbb{K}$-module of rank n, then an instance for $T^\mu V$ can be the tuple $(n, \mathbb{P}, g, \nu, \mu)$ with $\mathbb{P} \subseteq \langle v_\infty, v_\emptyset \rangle$ specifying the presence of the projective basis and $g : V \times V \to \mathbb{K}$ is a metric tensor specification. The type `TensorBundle{n,ℙ,g,ν,μ}` encodes this information as *byte-encoded* data available at pre-compilation, where μ is an integer specifying the order of the tangent bundle (i.e. multiplicity limit of the Leibniz-Taylor monomials). Lastly, ν is the number of tangent variables.

$$\langle v_1, \ldots, v_{n-\nu}, \partial_1, \ldots, \partial_\nu \rangle = M \leftrightarrow M' = \langle w_1, \ldots, w_{n-\nu}, \epsilon_1, \ldots, \epsilon_\nu \rangle$$

where v_i and w_i are bases for the vectors and covectors, while ∂_i and ϵ_j are bases for differential operators and scalar functions. The purpose of the `TensorBundle` type is to specify the $\mathbb{K}$-module basis at compile time. When assigned in a workspace, `V = Submanifold(::TensorBundle)` is used.

The metric signature of the `Submanifold{V,1}` elements of a vector space V can be specified with the `V"..."` by using $+$ or $-$ to specify whether the `Submanifold{V,1}` element of the corresponding index squares to $+1$ or -1. For example, `S"+++"` constructs a positive definite 3-dimensional `TensorBundle`, so constructors such as `S"..."` and `D"..."` are convenient. It is also possible to change the diagonal scaling, such as with `D"1,1,1,0"`, although the `Signature` format has a more compact representation if limited to $+1$ and -1. It is also possible to change the diagonal scaling, such as with `D"0.3,2.4,1"`. Fully general `MetricTensor` as a type with non-diagonal components requires a matrix, e.g. `MetricTensor([1 2; 3 4])`.

Declaring an additional point at infinity is done by specifying it in the string constructor with ∞ at the first index (i.e. Riemann sphere `S"∞+++"`). The hyperbolic geometry can be declared by $\emptyset$ subsequently (i.e. hyperbolic projection `S"∅+++"`). Additionally, the *null-basis* based on the projective split for conformal geometric algebra would be specified with `S"∞∅+++"`. These two declared basis elements are interpreted in the type system. The `tangent(V,`μ`,`ν`)` map can be used to specify μ and ν.

To assign `V = Submanifold(::TensorBundle)` along with associated basis elements of the `DirectSum.Basis` to the local Julia session workspace, it is typical to use `Submanifold` elements created by the `@basis` macro,

```
julia> using Grassmann; @basis S"-++" # macro or basis"-++"
(⟨-++⟩, v, v1, v2, v3, v12, v13, v23, v123)
```

the macro `@basis V` delcares a local basis in Julia. As a result of this macro, all `Submanifold{V,G}` elements generated with `M::TensorBundle` become available in the local workspace with the specified naming arguments. The first argument provides signature specifications, the second argument is the variable name for V the $\mathbb{K}$-module, and the third and fourth argument are prefixes of the `Submanifold` vector names (and covector names). Default is V assigned `Submanifold{M}` and v is prefix for the `Submanifold{V}`.

It is entirely possible to assign multiple different bases having different signatures without any problems. The `@basis` macro arguments are used to assign the vector space name to V and the basis elements to v_i, but other assigned names can be chosen so that their local names don't interfere: If it is undesirable to assign these variables to a local workspace, the versatile constructs of `DirectSum.Basis{V}` can be used to contain or access them, which is exported to the user as the method `DirectSum.Basis(V)`.

```
julia> DirectSum.Basis(V)
DirectSum.Basis{⟨-++⟩, 8}(v, v1, v2, v3, v12, v13, v23, v123)
```

`V(::Int...)` provides a convenient way to define a `Submanifold` by using integer indices to reference specific direct sums within the ambient space V.

Grassmann's exterior product is an anti-symmetric tensor product

$$v_i \wedge v_j = -v_j \wedge v_i \implies v_i \wedge v_i = 0 = -v_i \wedge v_i, \tag{0.1}$$

which generalizes the multilinear determinant transposition property

$$v_{\omega_1} \wedge \cdots \wedge v_{\omega_m} \wedge v_{\eta_1} \wedge \cdots \wedge v_{\eta_n} = (-1)^{mn} v_{\eta_1} \wedge \cdots \wedge v_{\eta_n} \wedge v_{\omega_1} \wedge \cdots \wedge v_{\omega_m}.$$

Hence for graded elements it is possible to deduce that

$$\omega \in \Lambda^m V, \quad \eta \in \Lambda^n V : \qquad \omega \wedge \eta = (-1)^{mn} \eta \wedge \omega. \tag{0.2}$$

Remark 0.11. Observe the anti-symmetry property implies that $\omega \otimes \omega = 0$, while the symmetric property neither implies nor denies such a property.

Example 0.12. Case of 3rd order tangent bundle operators composition:

$$T^3(\Lambda^0 V) = \partial_\emptyset \oplus \partial_1 \oplus \partial_2 \oplus \partial_3 \oplus (\partial_1 \circ \partial_2) \oplus (\partial_1 \circ \partial_3) \oplus (\partial_2 \circ \partial_3) \oplus (\partial_1 \circ \partial_2 \circ \partial_3)$$

In order to shorten the notation, the operation symbol is left out:

$$\{v_1, v_2, v_3, v_{12}, v_{13}, v_{23}, v_{123}\}, \{\partial_1, \partial_2, \partial_3, \partial_{12}, \partial_{13}, \partial_{23}, \partial_{123}\}$$

The canonical choice of orientation is with indices in sorted order, so that for example anti-symmetry is applied to rewrite $v_{21} = -v_{12}$ or the property $\partial_2 \circ \partial_1 = \partial_1 \circ \partial_2$ is applied for differential operators. In general, permutations of the indices get rendered as orientations of $(-1)^k$ of a basis $\mathbb{K}$-module.

Definition 0.13 (Permutations)**.** Consider $\sigma_j(\omega) = \sum_{k=0}^{n} (-1)^{\binom{k}{2^{j-1}}} \langle \omega \rangle_k$,

$$\sigma_1(\omega) \equiv \overline{\omega}, \qquad \sigma_2(\omega) \equiv \widetilde{\omega}, \qquad \sigma_{12} = \sigma_2(\sigma_1(\omega)) \equiv \widetilde{\overline{\omega}}$$

Theorem 0.14 ($\mathfrak{S}_j = \langle \sigma_1, \sigma_2, \ldots, \sigma_j \rangle$ is a group)**.** $\mathfrak{S}_2 = \{1, \sigma_1, \sigma_2, \sigma_{12}\}$ *is a set of automorphisms: grade involution* $\overline{\omega} = \sigma_1(\omega) = \sum_{k=0}^{n} (-1)^{\binom{k}{1}} \langle \omega \rangle_k$, *reverse* $\widetilde{\omega} = \sigma_2(\omega) = \sum_{k=0}^{n} (-1)^{\binom{k}{2}} \langle \omega \rangle_k = \sum_{k=0}^{n} (-1)^{(k-1)k/2} \langle \omega \rangle_k$ *is an anti-automorphism with* $\sigma_2(v_i \wedge v_j) = \sigma_2(v_j) \wedge \sigma_2(v_i)$*, and Clifford conjugate* $\widetilde{\overline{\omega}}$ *is the composition of grade involution and reverse anti-automorphism.*

Definition 0.15 (Real $\widetilde{\mathfrak{R}}\omega = (\omega + \widetilde{\omega})/2$ and imaginary $\widetilde{\mathfrak{I}}\omega = (\omega - \widetilde{\omega})/2$)**.** Real and imaginary define $\mathbb{Z}_2$-grading projections so that $\Lambda V = \widetilde{\mathfrak{R}}\Lambda V \oplus \widetilde{\mathfrak{I}}\Lambda V$; where $\widetilde{\mathfrak{R}}\Lambda V$ is the `real` part and $\widetilde{\mathfrak{I}}\Lambda V$ is the `imag` (imaginary) part.

Definition 0.16 (Even $\overline{\mathfrak{R}}\omega = (\omega + \overline{\omega})/2$ and odd $\overline{\mathfrak{I}}\omega = (\omega - \overline{\omega})/2$)**.** Even and odd define $\mathbb{Z}_2$-grading projections so that $\Lambda V = \overline{\mathfrak{R}}\Lambda V \oplus \overline{\mathfrak{I}}\Lambda V$; where $\overline{\mathfrak{R}}\Lambda V$ is the `even` part and $\overline{\mathfrak{I}}\Lambda V$ is the `odd` part.

In general, this can be extended to $\mathbb{Z}_2$-grading projections σ_j and its real $\sigma_j(\mathfrak{R})\omega = (\omega + \sigma_j(\omega))/2$ and imaginary $\sigma_j(\mathfrak{I})\omega = (\omega - \sigma_j(\omega))/2$ parts.

Grassmann.jl is a foundation which has been built up from a minimal $\mathbb{K}$-module algebra kernel on which an entirely custom algbera specification is designed and built from scratch on the base Julia language.

Definition 0.17. `TensorAlgebra{V,`$\mathbb{K}$`}` where `V::Submanifold{M}` for a generating $\mathbb{K}$-module specified by a `M::TensorBundle` choice

- `TensorBundle` specifies generators of `DirectSum.Basis` algebra
  - `Int` value induces a Euclidean metric of counted dimension
  - `Signature` uses `S"..."` with + and - specifying the metric
  - `DiagonalForm` uses `D"..."` for defining any diagonal metric
  - `MetricTensor` can accept non-diagonal metric tensor array
- `TensorGraded{V,G,`$\mathbb{K}$`}` has `grade` G and element of $\Lambda^G V$ subspace
  - `Chain{V,G,`$\mathbb{K}$`}` has a complete basis for $\Lambda^G V$ with $\mathbb{K}$-module
  - `Simplex{V}` alias column-module `Chain{V,1,Chain{V,1,`$\mathbb{K}$`}}`
- `TensorTerm{V,G,`$\mathbb{K}$`} <: TensorGraded{V,G,`$\mathbb{K}$`}` single coefficient
  - `Zero{V}` is a zero value which preserves V in its algebra type
  - `Submanifold{V,G,B}` $\langle v_{i_1} \wedge \cdots \wedge v_{i_G} \rangle_G$ with sorted indices B
  - `Single{V,G,B,`$\mathbb{K}$`}` where `B::Submanifold{V}` is paired to $\mathbb{K}$
- `AbstractSpinor{V,`$\mathbb{K}$`}` subtypes are special sub-algebras of ΛV
  - `Couple{V,B,`$\mathbb{K}$`}` is the sum of $\mathbb{K}$ scalar with `Single{V,G,B,`$\mathbb{K}$`}`
  - `PseudoCouple{V,B,`$\mathbb{K}$`}` is pseudoscalar + `Single{V,G,B,`$\mathbb{K}$`}`
  - `Spinor{V,`$\mathbb{K}$`}` has complete basis for the `even` $\mathbb{Z}_2$-graded terms
  - `CoSpinor{V,`$\mathbb{K}$`}` has complete basis for `odd` $\mathbb{Z}_2$-graded terms
- `Multivector{V,`$\mathbb{K}$`}` has complete basis for all ΛV with $\mathbb{K}$-module

Definition 0.18. `TensorNested{V,T}` subtypes are linear transformations

- `TensorOperator{V,W,T}` linear map $V \to W$ with `T::DataType`
  - `Endomorphism{V,T}` linear map $V \to V$ with `T::DataType`
- `DiagonalOperator{V,T}` diagonal map $V \to V$ with `T::DataType`
  - `DiagonalMorphism{V,<:Chain{V,1}}` diagonal map $V \to V$
  - `DiagonalOutermorphism{V,<:Multivector{V}}` : $\Lambda V \to \Lambda V$
- `Outermorphism{V,T}` extends $F \in$ `Endomorphism{V}` to full ΛV

$$F(v_1) \wedge \cdots \wedge F(v_n) = F(v_1 \wedge \cdots \wedge v_n) \tag{0.3}$$

- `Projector{V,T}` linear map $F : V \to V$ with $F(F) = F$ defined

$$\texttt{Proj(x::TensorGraded)} = \frac{x}{|x|} \otimes \frac{x}{|x|}$$

- `Dyadic{V,X,Y}` linear map $V \to V$ with `Dyadic(x,y)` $= x \otimes y$

Grassmann.jl was first to define a comprehensive `TensorAlgebra{V}` type system from scratch around the idea of the `V::Submanifold{M}` value to express algebra subtypes for a specified $\mathbb{K}$-module structure.

Accessing `metrictensor(V)` produces a linear map $g : V \to V$ which can be extended to $\Lambda g : \Lambda V \to \Lambda V$ outermorphism given by `metricextensor`. To apply the `metricextensor` to any `Grassmann` element of ΛV, the function `metric` can be used on the element, `cometric` applies a complement metric.

0.2. Grassmann-Hodge complement

John Browne has discussed the Grassmann duality principle, stating that every theorem (involving either of the exterior and regressive products) can be translated into its dual theorem by replacing the $\wedge$ and $\vee$ operations and applying Grassmann complements!

Definition 0.19 (Grassmann ! complement)**.** Expressed as unary operator, "right hand rule" is derived from John Browne's common factor theorem, given a pseudoscalar $\langle v_1 \wedge \cdots \wedge v_n \rangle_n \in \Lambda^n V$ the linear map $! : \Lambda^m V \to \Lambda^{n-m} V$

$$\langle v_{i_1} \wedge \cdots \wedge v_{i_m} \rangle_m \quad \mapsto \quad (-1)^{\frac{m(m+1)}{2} + \sum_{j=1}^{m} i_j} \langle \bigwedge_{k \neq i_j} v_k \rangle_{n-m}, \tag{0.4}$$

also denoted as `complementright` for "right hand rule". While the linear inverse of ! is a similar map `complementleft` expressible by re-orientation:

$$\langle v_{i_1} \wedge \cdots \wedge v_{i_m} \rangle_m \quad \mapsto \quad (-1)^{m(n-1)} ! \langle v_{i_1} \wedge \cdots \wedge v_{i_m} \rangle_m. \tag{0.5}$$

Together, these form an orthocomplementary propositional lattice $!, \wedge, \vee$

$$(! \bigvee_k \omega_k)(v_1, \ldots, v_n) = (\bigwedge_k ! \omega_k)(v_1, \ldots, v_n) \quad DeMorgan's\ Law, \tag{0.6}$$

where the regressive product $\vee$ satisfies the Grassmann laws with ! and $\wedge$.

Definition 0.20 (Hodge $\star$ complement)**.** Expressed as unary operator $\star$, define the composition of $\star =$ `complementright` $\circ$ `metric` as linear operator.

$$\star = ! \Lambda g : \Lambda V \to \Lambda V \tag{0.7}$$

This linear operator is also called `complementrighthodge` or only `hodge`.

Remark 0.21. Original Grassmann complement is equivalent to the Hodge complement with a Euclidean metric tensor, making `metric` an `identity`.

Definition 0.22. The interior contraction $\eta \cdot \omega = \eta \vee \star \omega$ is defined in terms of the regressive product and also the Hodge complement. By default the right contraction $>$ is used, but there is also a left contraction $<$ with swapped arguments $\eta < \omega = \omega \vee \star \eta$, and also $\eta >> \omega = \widetilde{\eta} > \omega$ with $\eta << \omega = \eta < \widetilde{\omega}$.

Remark 0.23. Using coupled subspaces in the block matrix structure of metric tensors, a basis element can be factorized in a corresponding way. In particular, for the diagonal metric this is simply the basis index factorization. However, a non-diagonal metric induces a more complex block factorization.

Definition 0.24 (Clifford geometric product)**.** If a_i is an indecomposable basis element with regards to the block matrix structure of the metric tensor and $B \in \Lambda^k V$ is a graded element, then define operation $\ominus$ as either

$$a_i \ominus B = a_i \wedge B + a_i < \widetilde{B}, \qquad B \ominus a_i = B \wedge a_i + \widetilde{B} > a_i \tag{0.8}$$

If $A = a_1 \wedge \cdots \wedge a_m$ are a basis factorization, then $a_1 \ominus \cdots \ominus a_m = a_1 \wedge \cdots \wedge a_m$. Furthermore, $a_1 \ominus \cdots \ominus (a_m \ominus B) = a_1 \ominus \cdots \ominus (a_m \wedge B + a_m < \widetilde{B})$ can be expanded to distribute the operations of $A \ominus B$. By applying this principle with the distributive law over the basis of ΛV, the Clifford product is defined. In Julia, the multiplication symbol $*$ can be used for geometric products.

Remark 0.25. For any $v_i \in \Lambda^1 V$, we define $v_i^2 = v_i v_i = g_{ii}$, so typically the diagonal metric g of the algebra is often defined by relations like these.

Definition 0.26 (Null-basis of projective split)**.** Let $v_\pm^2 = \pm 1$ be a basis with $v_\infty = v_+ + v_-$ and $v_\emptyset = (v_- - v_+)/2$. then $v_\infty^2 = 0, v_\emptyset^2 = 0, v_\infty \cdot v_\emptyset = -1$, and $v_{\infty\emptyset}^2 = 1$ with Lobachevskian plane $v_{\infty\emptyset}$ having these product properties:

$$\begin{aligned} v_{\infty\emptyset} v_\infty &= -v_\infty, & v_{\infty\emptyset} v_\emptyset &= v_\emptyset, \\ v_\infty v_\emptyset &= -1 + v_{\infty\emptyset}, & v_\emptyset v_\infty &= -1 - v_{\infty\emptyset}. \end{aligned}$$

Definition 0.27. The geometric product can be applied in two averaging operations, which are symmetrization and anti-symmetrization operations:

$$\bigodot_{k=1}^{j} \omega_k = \frac{1}{j!} \sum_{\sigma \in S_j} \prod \omega_{\sigma(k)}, \qquad \bigwedge_{k=1}^{j} \omega_k = \sum_{\sigma \in S_j} \frac{(-1)^{\varepsilon(\sigma)}}{j!} \prod_k \omega_{\sigma(k)} \tag{0.9}$$

Definition 0.28 (Reversed product)**.** Consider the reversed product $\langle \widetilde{\omega} \omega \rangle$.

$$|\omega|^2 = \langle \widetilde{\omega} \omega \rangle, \qquad |\omega| = \sqrt{\langle \widetilde{\omega} \omega \rangle}, \qquad ||\omega|| = \text{Euclidean } |\omega|.$$

Remark 0.29. In general $\sqrt{\omega} = e^{(\log \omega)/2}$ is valid for invertible ω.

Example 0.30 (Inverse)**.** A simple way to calculate algebraic inverses is $\omega^{-1} = \widetilde{\omega}(\widetilde{\omega}\omega)^{-1} = \widetilde{\omega}/|\omega|^2$, with $\eta/\omega = \eta\omega^{-1}$ and $\eta \backslash \omega = \eta^{-1}\omega$.

Definition 0.31 (Sandwich product)**.** Define operator as $\eta \oslash \omega = \overline{\omega}^{-1} \eta \omega$. Alternatively, the reversed definition is $\eta \omega \overline{\eta}^{-1}$ typically notated η `>>>` ω.

Since $\langle (\widetilde{\omega} + \omega)(\omega + \widetilde{\omega}) \rangle = (\omega + \widetilde{\omega})^2$, it follows $|\Re\omega|^2 = (\Re\omega)^2$. Similarly, $\langle (\widetilde{\omega} - \omega)(\omega - \widetilde{\omega}) \rangle = -(\omega + \widetilde{\omega})^2$ implies $|\mathfrak{I}\omega|^2 = -(\mathfrak{I}\omega)^2$. Due to the $\mathbb{Z}_2$-grading induced by $\omega = \Re\omega + \mathfrak{I}\omega$, it is possible to partition real and imaginary by

$$\langle \widetilde{\omega} \rangle_r / \left| \langle \omega \rangle_r \right| = \sqrt{\langle \widetilde{\omega} \rangle_r^2 / \left| \langle \omega \rangle_r \right|^2} = \sqrt{\langle \widetilde{\omega} \rangle_r / \langle \omega \rangle_r} = \sqrt{(-1)^{(r-1)r/2}} \in \left\{ 1, \sqrt{-1} \right\},$$

which is a unique partitioning completely independent of the metric space and manifold of the algebra.

$$\widetilde{\omega}\omega = |\omega|^2 = |\Re\omega + \mathfrak{I}\omega|^2 = |\Re\omega|^2 + |\mathfrak{I}\omega|^2 + 2\Re(\Re\omega \mathfrak{I}\omega)$$

Lemma 0.32. *Let $\omega \in \Lambda^m V$, then $I \vee \omega = \omega$.*

Proof. $I \vee \omega = !^{-1}(!I \wedge !\omega) = !^{-1}(1 \wedge !\omega) = !^{-1}!\omega = \omega$. □

Corollary 0.33. Observe, $\star\omega = \widetilde{\omega} I = I \cdot \omega$ since $I \cdot \omega = I \vee \star\omega = \star\omega$.

Theorem 0.34. *Let $\omega \in \Lambda^m V$, then $\star\star\omega = (-1)^{m(n-m)}\omega|I|^2$.*

Proof. Let $\omega \in \omega^m V$, note that $\star\omega = \widetilde{\omega} I$, then rewrite the expressions

$$\begin{aligned}\star\star\omega &= \widetilde{\widetilde{\omega} I} I = (-1)^{m(m-1)/2}\widetilde{\omega I} I \\ &= (-1)^{m(m-1)/2}(-1)^{(n-m-1)(n-m)/2}\omega I^2 \\ &= (-1)^{m(n-m)}(-1)^{n(n-1)/2}\omega I^2 \\ &= (-1)^{m(n-m)}\omega \widetilde{I} I = (-1)^{m(n-m)}\omega|I|^2\end{aligned}$$

Hence, the result follows since $(-1)^{n(n-1)/2} I I = \widetilde{I} I = I \cdot I = |I|^2$. □

Corollary 0.35 (Euclidean complement of a complement)**.** Let $\omega \in \Lambda^m(\mathbb{R}^n)$ then $\star\star\omega = (-1)^{m(n-m)}\omega$ since $|I|^2 = 1$.

Theorem 0.36. *Let $\omega \in \Lambda^m V$, then $(\omega \vee \star\omega) I = \omega \wedge \star\omega$*

Proof. It is straight forward to check based on properties of $!, \star, \wedge, \vee$ that

$$\begin{aligned}\omega \vee \star\omega = \omega \vee (\widetilde{\omega} I) &= !^{-1}(!\omega \wedge !(\widetilde{\omega} I)) \\ &= g(\omega, I)(-1)^{m(n-m)}!^{-1}(!\omega \wedge \omega) \\ &= g(\omega, I)!^{-1}(\omega \wedge !\omega) = (\omega \wedge \star\omega)/I\end{aligned}$$

From this the result follows, after multiplying by I pseudoscalar. □

Theorem 0.37. $\eta \wedge \star\omega = (\widetilde{\omega} \vee \star\widetilde{\eta}) I = (\widetilde{\omega} \cdot \widetilde{\eta}) I \iff \eta \cdot \omega = \eta \vee \star\omega = (\widetilde{\omega} \wedge \star\widetilde{\eta})/I$.

Theorem 0.38. *Let $\eta, \omega \in \Lambda^m V$, then $\widetilde{\eta} \cdot \widetilde{\omega}) = \eta \cdot \omega$.*

Proof. Let $\eta, \omega \in \Lambda^m V$, then $\widetilde{\eta} \cdot \widetilde{\omega} = ((-1)^{m(n-m)})^2(\eta \cdot \omega) = \eta \cdot \omega$. □

Corollary 0.39 (Absolute value $|\omega|^2 = \omega \cdot \omega$)**.**

$$(\omega \cdot \omega) I = \widetilde{\omega} \wedge \star\widetilde{\omega} = \widetilde{\omega} \star \widetilde{\omega} = \widetilde{\omega}\omega I = |\omega|^2 I \iff \omega \cdot \omega = \widetilde{\omega}\omega$$

Theorem 0.40 (Hodge complement)**.** *Let $\omega \in \Lambda^m V$, then $\omega \wedge \star\omega = \langle \omega \vee \star\omega \rangle I$*

Proof. Observe that $\omega \wedge \star\omega = \omega \star \omega = \omega\widetilde{\omega} I = |\omega|^2 I = \langle \omega > \omega \rangle I$. □

Grassmann's (graded tensor) right contraction is written $\eta > \omega = \eta \vee \star\omega$,

Contraction	left(η, ω)	right(η, ω)
Grassmann	$\langle\eta\rangle_r < \langle\omega\rangle_s = \langle\widetilde{\eta}\omega\rangle_{s-r}$	$\langle\eta\rangle_r > \langle\omega\rangle_s = \langle\widetilde{\eta}\omega\rangle_{r-s}$
Reversed	$\langle\widetilde{\eta}\rangle_r < \langle\widetilde{\omega}\rangle_s = \langle\eta\widetilde{\omega}\rangle_{s-r}$	$\langle\widetilde{\eta}\rangle_r > \langle\widetilde{\omega}\rangle_s = \langle\eta\widetilde{\omega}\rangle_{r-s}$
Conventional	$\langle\eta\rangle_r < \langle\widetilde{\omega}\rangle_s = \langle\eta\omega\rangle_{s-r}$	$\langle\widetilde{\eta}\rangle_r > \langle\omega\rangle_s = \langle\eta\omega\rangle_{r-s}$

Definition 0.41. Common unary operations on `TensorAlgebra` elements

- `Manifold` returns the parameter `V::Submanifold{M}` $\mathbb{K}$-module
- `mdims` dimensionality of the pseudoscalar V of that `TensorAlgebra`
- `gdims` dimensionality of the grade G of V for that `TensorAlgebra`
- `tdims` dimensionality of `Multivector{V}` for that `TensorAlgebra`
- `grade` returns G for `TensorGraded{V,G}` while `grade(x,g)` is $\langle x\rangle_g$
- `istensor` returns true for `TensorAlgebra` elements
- `isgraded` returns true for `TensorGraded` elements
- `isterm` returns true for `TensorTerm` elements
- `complementright` Euclidean metric Grassmann right complement
- `complementleft` Euclidean metric Grassmann left complement
- `complementrighthodge` Grassmann-Hodge right complement $\widetilde{\omega}I$
- `complementlefthodge` Grassmann-Hodge left complement $I\widetilde{\omega}$
- `metric` applies the `metricextensor` as outermorphism operator
- `cometric` applies complement `metricextensor` as outermorphism
- `metrictensor` returns $g : V \to V$ associated to `TensorAlgebra{V}`
- `metrictextensor` returns $\Lambda g : \Lambda V \to \Lambda V$ for `TensorAlgebra{V}`
- `involute` grade permutes basis with $\langle\overline{\omega}\rangle_k = \sigma_1(\langle\omega\rangle_k) = (-1)^k\langle\omega\rangle_k$
- `reverse` permutes basis with $\langle\widetilde{\omega}\rangle_k = \sigma_2(\langle\omega\rangle_k) = (-1)^{k(k-1)/2}\langle\omega\rangle_k$
- `clifford` conjugate of an element is composite `involute` $\circ$ `reverse`
- `even` part selects $\overline{\Re}\omega = (\omega + \overline{\omega})/2$ and is defined by Λ^g for even g
- `odd` part selects $\overline{\Im}\omega = (\omega - \overline{\omega})/2$ and is defined by Λ^g for odd g
- `real` part selects $\widetilde{\Re}\omega = (\omega + \widetilde{\omega})/2$ and is defined by $|\widetilde{\Re}\omega|^2 = (\widetilde{\Re}\omega)^2$
- `imag` part selects $\widetilde{\Im}\omega = (\omega - \widetilde{\omega})/2$ and is defined by $|\widetilde{\Im}\omega|^2 = -(\widetilde{\Im}\omega)^2$
- `abs` is the absolute value $|\omega| = \sqrt{\widetilde{\omega}\omega}$ and `abs2` is then $|\omega|^2 = \widetilde{\omega}\omega$
- `norm` evaluates a positive definite norm metric on the coefficients
- `unit` applies normalization defined as `unit(t) = t/abs(t)`
- `scalar` selects grade 0 term of any `TensorAlgebra` element
- `vector` selects grade 1 terms of any `TensorAlgebra` element
- `bivector` selects grade 2 terms of any `TensorAlgebra` element
- `trivector` selects grade 3 terms of any `TensorAlgebra` element
- `pseudoscalar` max. grade term of any `TensorAlgebra` element
- `value` returns internal `Values` tuple of a `TensorAlgebra` element
- `valuetype` returns type of a `TensorAlgebra` element value's tuple

Binary operations commonly used in `Grassmann` algebra syntax

- `+` and `-` carry over from the $\mathbb{K}$-module structure associated to $\mathbb{K}$
- `wedge` is exterior product $\wedge$ and `vee` is regressive product $\vee$
- `>` is the right contraction and `<` is the left contraction for ΛV
- `*` is the geometric product and `/` uses `inv` algorithm for division
- $\oslash$ is the `sandwich` and `>>>` is its alternate operator orientation

Custom methods related to tensor operators and roots of polynomials

- `inv` returns the inverse and `adjugate` returns transposed cofactor
- `det` returns the scalar determinant of an endomorphism operator
- `tr` returns the scalar trace of an endomorphism operator
- `transpose` operator has swapping of row and column indices
- `compound(F,g)` is multilinear endomorphism $\Lambda^g F : \Lambda^g V \to \Lambda^g V$
- `outermorphism(A)` transforms $A : V \to V$ into $\Lambda A : \Lambda V \to \Lambda V$
- `operator` make linear representation of multivector outermorphism
- `companion` matrix of monic polynomial $a_0 + a_1 z + \cdots + a_n z^n + z^{n+1}$
- `roots(a...)` of polynomial with coefficients $a_0 + a_1 z + \cdots + a_n z^n$
- `rootsreal` of polynomial with coefficients $a_0 + a_1 z + \cdots + a_n z^n$
- `rootscomplex` of polynomial with coefficients $a_0 + a_1 z + \cdots + a_n z^n$
- `monicroots(a...)` of monic polynomial $a_0 + a_1 z + \cdots + a_n z^n + z^{n+1}$
- `monicrootsreal` of monic polynomial $a_0 + a_1 z + \cdots + a_n z^n + z^{n+1}$
- `monicrootscomplex` of monic polynomial $a_0 + a_1 z + \cdots + a_n z^n + z^{n+1}$
- `characteristic(A)` polynomial coefficients from $\det(A - \lambda I)$
- `eigvals(A)` are the eigenvalues $[\lambda_1, \ldots, \lambda_n]$ so that $Ae_i = \lambda_i e_i$
- `eigvalsreal` are real eigenvalues $[\lambda_1, \ldots, \lambda_n]$ so that $Ae_i = \lambda_i e_i$
- `eigvalscomplex` are complex eigenvalues $[\lambda_1, \ldots, \lambda_n]$ so $Ae_i = \lambda_i e_i$
- `eigvecs(A)` are the eigenvectors $[e_1, \ldots, e_n]$ so that $Ae_i = \lambda_i e_i$
- `eigvecsreal` are real eigenvectors $[e_1, \ldots, e_n]$ so that $Ae_i = \lambda_i e_i$
- `eigvecscomplex` are complex eigenvectors $[e_1, \ldots, e_n]$ so $Ae_i = \lambda_i e_i$
- `eigen(A)` spectral decomposition $\sum_i \lambda_i \mathrm{Proj}(e_i)$ with $Ae_i = \lambda_i e_i$
- `eigenreal` spectral decomposition $\sum_i \lambda_i \mathrm{Proj}(e_i)$ with $Ae_i = \lambda_i e_i$
- `eigencomplex` spectral decomposition $\sum_i \lambda_i \mathrm{Proj}(e_i)$ so $Ae_i = \lambda_i e_i$
- `eigpolys(A)` normalized symmetrized functions of `eigvals(A)`
- `vandermonde` facilitates $((X'X)^{-1}X')y$ for polynomial coefficients
- `cayley(V,∘)` returns product table for V and binary operation $\circ$

0.3. Tensor field topology and fiber bundles

Definition 0.42. Commonly used fundamental building blocks are

- `ProductSpace{V,𝕂,N} <: AbstractArray{Chain{V,1,𝕂,N},N}` uses Cartesian products of interval subsets of $\mathbb{R}\times\mathbb{R}\times\cdots\times\mathbb{R} = \Lambda^1\mathbb{R}^n$, generates lazy array of `Chain{V,1}` point vectors from input ranges
  - `ProductSpace{V}(0:0.1:1,0:0.1:1) # specify V`
  - `ProductSpace(0:0.1:1,0:0.1:1) # auto-select V`
  - `ProductSpace{V}(r::AbstractVector{<:Real}...)` default
  - `⊕(r::AbstractVector{<:Real}...)` for algebraic syntax
- `Global{N,T}` represents array with same T value at all indices
- `LocalFiber{B,F}` has a local `basetype` of B and `fibertype` of F
  - `Coordinate{P,G}` has `pointtype` of P and `metrictype` of G
- `ImmersedTopology{N,M} = AbstractArray{Values{N,Int},M}`
  - `ProductTopology` generates basic product topologies for grids
  - `SimplexTopology` defines continuous simplex immersion
  - `DiscontinuousTopology` disconnects for discontinuous
  - `LagrangeTopology` extends for Lagrange polynomial base
  - `QuotientTopology` defines classes of quotient identification

Generalizing upon `ProductTopology`, the `QuotientTopology` defines a quotient identification across the boundary fluxes of the region, which then enables different specializations of `QuotientTopology` as

- `OpenTopology`: all boundaries don't have accumulation points,
- `CompactTopology`: all points have a neighborhood topology,
- `CylinderTopology`: closed ribbon with two edge open endings,
- `MobiusTopology`: twisted ribbon with one edge open ending,
- `WingTopology`: upper and lower surface topology of wing,
- `MirrorTopology`: reflection boundary along mirror (basis) axis,
- `ClampedTopology`: each boundary face is reflected to be compact,
- `TorusTopology`: generalized compact torus up to 5 dimensions,
- `HopfTopology`: compact topology of the Hopf fibration in 3D,
- `KleinTopology`: compact topology of the Klein bottle domain,
- `PolarTopology`: polar compactification with open edge boundary,
- `SphereTopology`: generalized mathematical sphere, compactified,
- `GeographicTopology`: axis swapped from `SphereTopology` in 2D.

Combination of `PointArray <: Coordinates` and `ImmersedTopology` leads into definition of `TensorField` as a global section of a `FrameBundle`.

All these methods apply to `SimplexTopology` except `isopen`, `iscompact`

- `isopen` is true if `QuotientTopology` is an `OpenTopology` instance
- `iscompact` is true if `QuotientTopology` is a `CompactTopology`
- `nodes` counts number of `vertices` associated to `SimplexTopology`
- `sdims` counts the number of vertices N of a `SimplexTopology{N}`
- `subelements` subspace element indices associated to `fulltopology`
- `subimmersion` modified with vertices re-indexed based on subspace
- `topology` view into `fulltopology` based on `subelements` structure
- `totalelements` counts total number of elements in `fulltopology`
- `totalnodes` counts total number of nodes over all subspaces
- `vertices` list of indices associated to the subspace `immersion`
- `elements` counts the number of `subelements` in the `immersion`
- `isfull` is true if the `immersion` is a `fulltopology`, not a subspace
- `istotal` is true if `fulltopology` is covering `totalnodes` completely
- `iscover` is true if `isfull` and `istotal`, fully covering `totalnodes`
- `isdiscontinuous` is true if an instance of `DiscontinuousTopology`
- `isdisconnected` is true if `isdiscontinuous` and fully disconnected
- `continuous` returns original data from `DiscontinuousTopology`
- `discontinuous` allocates a derived `DiscontinuousTopology`
- `disconnect` allocates a disconnected `DiscontinuousTopology`
- `getfacet` indexing `subelements` in reference to the `fullimmersion`
- `getimage` indexing vertex subspace in reference to `fullimmersion`
- `edges` assembles `SimplexTopology{2}` of all unique edge elements
- `facets` assembles `SimplexTopology` of all unique facet elements
- `complement` returns a `SimplexTopology` based on `subelements`
- `interior` returns the interior components of a `SimplexTopology`
- ∂ operator returns boundary components of a `SimplexTopology`
- `degrees` returns the (graph) degree for each incidence vertex
- `weights` divides each incidence vertex by the (graph) degree
- `adjacency` returns a symmetric sparse matrix with ones at vertices
- `antiadjacency` returns sparse matrix with vertex antisymmetry
- `incidence` returns heterogenous relation for vertices and elements
- `laplacian` returns the (graph) Laplacian with adjacent vertices
- `neighbors` finds neighboring elements per `SimplexTopology` facet

Definition 0.43. An n-dimensional *manifold* M requires the existence of a neighborhood U for each $p \in U \subseteq M$ with a local *chart* map $\phi : U_\phi \to \mathbb{R}^n$. Given another chart $\psi : U_\psi \to \mathbb{R}^n$, then the combinatorial compositions

$$\phi \circ \psi^{-1} : \psi(U_\phi \cap U_\psi) \to \phi(U_\phi \cap U_\psi), \quad \psi \circ \phi^{-1} : \phi(U_\phi \cap U_\psi) \to \psi(U_\phi \cap U_\psi)$$

are the transition maps. If all the transition maps ϕ are $\mathcal{C}^r$ differentiable and $\bigcup_\phi U_\phi = M$, then the collection of charts is called an *atlas* of a $\mathcal{C}^r$ manifold.

Definition 0.44. A *fiber bundle* is a manifold E with projection $\pi : E \to B$ commutes with local trivializations ϕ paired to U_ϕ of manifold $B = \bigcup_\phi U_\phi$

$$\begin{array}{ccc} \pi^{-1}(U_\phi) & \xrightarrow{\phi} & U_\phi \times F \\ & \searrow^{\pi} & \downarrow{\scriptstyle \pi_1} \\ & & U_\phi \end{array} , \tag{0.10}$$

where B is the `basetype` and F is the `fibertype` of $E_p = \pi^{-1}(p) = \{p\} \times F$,

$$E = \bigcup_{p \in B} E_p = \bigcup_{p \in B} \{p\} \times F = B \times F, \qquad B = \bigcup_{\phi} U_\phi.$$

`FiberBundle{E,N} <: AbstractArray{E,N}` where E is the `eltype`

- `Coordinates{P,G,N} <: FiberBundle{Coordinate{P,G},N}`
  - `PointArray{P,G,N}` has `pointtype` of P, `metrictype` of G
  - `FiberProduct` introduces fiber product structure for manifolds
- `FrameBundle{C,N}` has `coordinatetype` of C and `immersion`
  - `GridBundle{N,C}` N-grid with `coordianates` and `immersion`
  - `SimplexBundle{N,C}` defines `coordinates` and an `immersion`
  - `FaceBundle{N,C}` defines `element` faces and their `immersion`
  - `FiberProductBundle` for extruding dimensions from simplices
  - `HomotopyBundle` encapsulates a variation as `FrameBundle`
- `TensorField` defines fibers in a global section of a `FrameBundle`

When a `TensorField` has a `fibertype` from $\Lambda^g TV$ then it is a grade g differential form on the tangent bundle of V. In general the `TensorField` type can deal with more abstract `fibertype` varieties than only those used for differential forms, as it unifies many different forms of tensor analysis.

By default, the `InducedMetric` is defined globally in each `PointArray`, unless a particular metric tensor specification is provided. When the default `InducedMetric` is invoked, the metric tensor from the `TensorAlgebra{V}` type is used for the global manifold, instead of the extra allocation to specify metric tensors at each point. `FrameBundle` then defines local charts along with metric tensor in a `PointArray` and pairs it with an `ImmersedTopology`. Then the fiber of a `FrameBundle` section is a fiber of a `TensorField`.

These methods relate to `FrameBundle` and `TensorField` instances

- `coordinates(m::FiberBundle)` returns `Coordinates` data type
- `coordinatetype` return applies to `FiberBundle` or `LocalFiber`
- `immersion(m::FiberBundle)` returns `ImmersedTopology` data
- `immersiontype` return applies to `FiberBundle` or `LocalFiber`
- `base` returns the B element of a `LocalFiber{B,F}` or `FiberBundle`
- `basetype` returns type B of a `LocalFiber{B,F}` or `FiberBundle`
- `fiber` returns the F element of `LocalFiber{B,F}` or `FiberBundle`
- `fibertype` returns the F type of `LocalFiber{B,F}` or `FiberBundle`
- `points` returns `AbstractArray{P}` data for `Coordinates{P,G}`
- `pointtype` is type P of `Coordinate{P,G}` or `Coordinates{P,G}`
- `metrictensor` returns the grade 1 block of the `metricextensor`
- `metricextensor` is `AbstractArray{G}` data for `Coordinates{P,G}`
- `metrictype` is type G of `Coordinate{P,G}` or `Coordinates{P,G}`
- `fullcoordinates` returns full `FiberBundle{Coordinate{P,G}}`
- `fullimmersion` returns superset `ImmersedTopology` which `isfull`
- `fulltopology` returns composition of `topology` $\circ$ `fullimmersion`
- `fullvertices` list of `vertices` associated to the `fullimmersion`
- `fullpoints` is full `AbstractArray{P}` instead of possibly subspace
- `fullmetricextensor` is full `AbstractArray{G}` instead of subspace
- `isinduced` is true if the `metrictype` is an `InducedMetric` type
- `bundle` returns the integer identification of bundle cache

Various interpolation methods are also supported and can be invoked by applying `TensorField` instances as function evaluations on base manifold or applying some form of resampling method to the manifold topology.

- `volumes` returns `FaceBundle` with simplex volume at each element
- `initmesh` provides API keyword for interfacing mesh initialization
- `refinemesh` provides API keyword for interfacing mesh refinement
- `affinehull` returns a localized affine simplex hull at each element
- `affineframe` returns a localized affine basis frame at each element
- `gradienthat` returns the hat gradients for the `SimplexBundle`

For `GridBundle` initialization it is typical to invoke a combination of `ProductSpace` and `QuotientTopology`, while optional Julia packages extend `SimplexBundle` initialization, such as `Meshes`, `GeometryBasics`, `Delaunay`, `QHull`, `MiniQhull`, `Triangulate`, `TetGen`, `MATLAB`, `FlowGeometry`.

Definition 0.45. Let $\gamma : [a,b] \to \mathbb{R}^n$ be a curve $\gamma(t)$ with parameter t.

- `integral(::IntervalMap)` cumulative trapezoidal sum $\int_a^t \gamma(\xi) d\xi$
- `integrate(::IntervalMap)` final value of $\int_a^b \gamma(t) dt$ on interval end
- `arclength(::IntervalMap)` curve parameter $s(t) = \int_a^t |\frac{d\gamma(\xi)}{d\xi}| d\xi$
- `tangent(::IntervalMap)` tangent speed curve $\frac{d\gamma(t)}{dt} = \frac{ds}{dt} T(t)$
- `unittangent(::IntervalMap)` unit tangent curve $T(t) = \frac{d\gamma}{dt}\frac{dt}{ds}$
- `speed(::IntervalMap)` tangent speed of a curve $\frac{ds}{dt} = \left|\frac{d\gamma(t)}{dt}\right|$
- `normal(::IntervalMap)` $\kappa(t)N(t) = \frac{dT}{dt}\frac{dt}{ds} = \frac{d}{dt}\left(\frac{d\gamma}{dt}\frac{dt}{ds}\right)\frac{dt}{ds}$
- `unitnormal(::IntervalMap)` $N(t) = \frac{dT}{dt}\frac{dt}{ds}/\kappa(t)$ normalized
- `curvature(::AbstractCurve)` $\kappa(t) = \left|\frac{dT}{dt}\frac{dt}{ds}\right| = \left|\frac{d}{dt}\left(\frac{d\gamma}{dt}\frac{dt}{ds}\right)\frac{dt}{ds}\right|$
- `radius(::AbstractCurve)` of curvature $\kappa(t)^{-1} = \left|\frac{d}{dt}\left(\frac{d\gamma}{dt}\frac{dt}{ds}\right)\frac{dt}{ds}\right|^{-1}$
- `evolute(::AbstractCurve)` $\gamma + \frac{N}{\kappa} = \gamma(t) + \frac{d}{dt}\left(\frac{d\gamma}{dt}\frac{dt}{ds}\right)\frac{dt}{ds}/(\kappa(t))^2$
- `involute(::AbstractCurve)` $\gamma - Ts = \gamma(t) - \left(\frac{d\gamma(t)}{dt}\frac{dt}{ds}\right)\int_a^t |\frac{d\gamma(\xi)}{d\xi}| d\xi$
- `osculatingplane(::AbstractCurve)` linear span of $\left[\frac{ds}{dt}T, \kappa N\right]$
- `unitosculatingplane(::AbstractCurve)` linear span of $[T, N]$
- `binormal(::SpaceCurve)` complement $\frac{ds}{dt}\kappa B = \star(\frac{ds}{dt}T \wedge \kappa N)$
- `unitbinormal(::SpaceCurve)` complement of plane $B = \star(T \wedge N)$
- `torsion(::SpaceCurve)` $\tau(t) = \left|\frac{dB}{dt}\frac{dt}{ds}\right| = \left|\frac{d}{dt} \star (T \wedge N)\frac{dt}{ds}\right|$
- `frame(::AbstractCurve)` scaled frame $\left[\frac{ds}{dt}T, \kappa N, \star(\frac{ds}{dt}T \wedge \kappa N)\right]$
- `unitframe(::AbstractCurve)` Frenet frame $[T, N, \star(T \wedge N)]$
- `curvatures(::AbstractCurve)` all degrees of freedom $[\kappa, \tau, \dots]$
- `curvatures(::AbstractCurve,i)` selects i-th curvature degree
- `bishopframe(::SpaceCurve,`θ_0`=0)` computes Bishop-style frame
- `bishopunitframe(::SpaceCurve,`θ_0`=0)` Bishop-style unit frame
- `bishoppolar(::SpaceCurve,`θ_0`=0)` Bishop polar $(\kappa, \theta_0 + \int_a^b \tau ds)$
- `bishop(::SpaceCurve,`θ_0`=0)` $\kappa(\cos(\theta_0 + \int_a^b \tau ds), \sin(\theta_0 + \int_a^b \tau ds))$
- `planecurve(::RealFunction,`θ_0`=0)` from curvature $\kappa(t)$ and θ_0

$$(\kappa(t), \theta_0) \mapsto \int_a^b \left[cos(\theta_0 + \int_a^b \kappa(t) dt), \sin(\theta_0 + \int_a^b \kappa(t) dt)\right] dt$$

- `linkmap(f::SpaceCurve,g::SpaceCurve)` is $\ell(t,s) = g(s) - f(t)$
- `linknumber(f,g)` of curves $\propto$ `sectorintegrate` $\circ$ `unit` $\circ$ `linkmap`

Definition 0.46. Surfaces $\gamma : \mathbb{R}^2 \to \mathbb{R}^3$ with parametric $\gamma(x_1, x_2)$ methods

- `graph` outputs surface $\gamma : \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}$ from scalar field $f : \mathbb{R}^n \to \mathbb{R}$
- `normal` vector $N(x) = \star(\frac{\partial\gamma(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n})$ or product $\frac{\partial\gamma(x)}{\partial x_1} \times \frac{\partial\gamma(x)}{\partial x_2}$
- `unitnormal` $\nu(x) = \star(\frac{\partial\gamma(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n}) / |\star(\frac{\partial\gamma(x)}{x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n})|$
- `normalnorm` is the norm of normal $|N(x)| = |\star(\frac{\partial\gamma(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n})|$
- `jacobian` linear span of $\left[\frac{\partial\gamma(x)}{\partial x_1}, \ldots, \frac{\partial\gamma(x)}{\partial x_n}\right]$ as `TensorOperator`
- `weingarten` linear span of $\left[\frac{\partial\nu(x)}{\partial x_1}, \ldots, \frac{\partial\nu(x)}{\partial x_n}\right]$ as `TensorOperator`
- `sectordet` is the product $\gamma(x) \wedge \frac{\partial\gamma(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n}$, here with $n = 2$
- `sectorintegral` $\int \frac{\gamma(x)}{n+1} \wedge \frac{\partial\gamma(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n} dx_1 \cdots dx_n$ with $n = 2$
- `sectorintegrate` estimates the total value of `sectorintegral`
- `area` cumulative $\int |\star(\frac{\partial\gamma(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma(x)}{\partial x_n})| dx_1 \cdots dx_n$ with $n = 2$
- `surfacearea` estimates total value of the (surface) `area` integral
- `surfacemetric` gets `GridBundle` with `firstform` as `metrictensor`
- `surfacemetricdiag` gets `GridBundle` with `firstformdiag` metric
- `surfaceframe` constructs intrinsic orthonormal surface frame
- `frame` scaled Darboux style frame $\left[\frac{\partial\gamma(x)}{\partial x_1}, \star\left(N(x) \wedge \frac{\partial\gamma(x)}{\partial x_1}\right), N(x)\right]$
- `unitframe` is then $\left[\frac{\partial\gamma(x)}{\partial x_1} / \left|\frac{\partial\gamma(x)}{\partial x_1}\right|, \star\left(\nu(x) \wedge \frac{\partial\gamma(x)}{\partial x_1}\right) / \left|\frac{\partial\gamma(x)}{\partial x_1}\right|, \nu(x)\right]$
- `firstform` $I = g = \begin{bmatrix} \frac{\partial\gamma(x)}{\partial x_1} \cdot \frac{\partial\gamma(x)}{\partial x_1} & \frac{\partial\gamma(x)}{\partial x_1} \cdot \frac{\partial\gamma(x)}{\partial x_2} \\ \frac{\partial\gamma(x)}{\partial x_1} \cdot \frac{\partial\gamma(x)}{\partial x_2} & \frac{\partial\gamma(x)}{\partial x_2} \cdot \frac{\partial\gamma(x)}{\partial x_2} \end{bmatrix}$ or `firstformdiag`
- `secondform` $II = \begin{bmatrix} \nu(x) \cdot \frac{\partial^2\gamma(x)}{\partial x_1^2} & \nu(x) \cdot \frac{\partial^2\gamma(x)}{\partial x_1 \partial x_2} \\ \nu(x) \cdot \frac{\partial^2\gamma(x)}{\partial x_1 \partial x_2} & \nu(x) \cdot \frac{\partial^2\gamma(x)}{\partial x_2^2} \end{bmatrix}$ 2nd fundamental
- `thirdform` III is the composition map `firstform` ∘ `unitnormal`
- `shape` is the geometry shape operator $I(x)^{-1} II(x)$ of a surface $\gamma(x)$
- `principals` (curvatures) are the composition `eigvals` ∘ `shape`
- `principalaxes` (curvatures) are the composition `eigvecs` ∘ `shape`
- `curvatures` (polynomials) are the composition `eigpolys` ∘ `shape`
- `curvatures(::TensorField,i)` selects i-th curvature polynomial
- `meancurvature` is the mean curvature (first curvature) of the `shape`
- `gaussintrinsic` is the (intrinsic) Gauss curvature (last curvature)
- `gaussextrinsic` is the (extrinsic) Gauss curvature in sector form
- `gausssign` is the sign of the Gauss curvature of the `shape`

`AbstractTensors`, `Grassmann` settled on custom trigonometric identities,

$$\exp(\omega) = \sum_{n=0}^{\infty} \frac{\omega^n}{n!}, \qquad \log(\omega) = \sum_{n=0}^{\infty} \frac{2}{2n+1}\left(\frac{\omega-1}{\omega+1}\right)^{2n+1}$$

$$\cosh(\omega) = \sum_{n=0}^{\infty} \frac{\omega^{2n}}{(2n)!}, \qquad \sinh(\omega) = \sum_{n=0}^{\infty} \frac{\omega^{2n+1}}{(2n+1)!},$$

$$\cos(\omega) = \cosh(I\omega), \qquad \sin(\omega) = \sinh(I\omega)/I,$$

$$\tan(\omega) = \frac{\sin(\omega)}{\cos(\omega)}, \qquad \cot(\omega) = \frac{\cos(\omega)}{\sin(\omega)},$$

$$\sec(\omega) = \frac{1}{\cos(\omega)}, \qquad \csc(\omega) = \frac{1}{\sin(\omega)},$$

$$\operatorname{asec}(\omega) = \operatorname{acos}(\omega^{-1}), \qquad \operatorname{acsc}(\omega) = \operatorname{asin}(\omega^{-1}),$$

$$\operatorname{sech}(\omega) = \frac{1}{\cosh(\omega)}, \qquad \operatorname{csch}(\omega) = \frac{1}{\sinh(\omega)},$$

$$\operatorname{asech}(\omega) = \operatorname{acosh}(\omega^{-1}), \qquad \operatorname{acsch}(\omega) = \operatorname{asinh}(\omega^{-1}),$$

$$\tanh(\omega) = \frac{\sinh(\omega)}{\cosh(\omega)}, \qquad \coth(\omega) = \frac{\cosh(\omega)}{\sinh(\omega)},$$

$$\operatorname{asinh}(\omega) = \log\left(\omega + \sqrt{\omega^2+1}\right), \qquad \operatorname{acosh}(\omega) = \log\left(\omega + \sqrt{\omega^2-1}\right),$$

$$\operatorname{atanh}(\omega) = \frac{\log(1+\omega) - \log(1-\omega)}{2}, \qquad \operatorname{acoth}(\omega) = \frac{\log(\omega+1) - \log(\omega-1)}{2},$$

$$\operatorname{asin}(\omega) = -I\log\left(I\omega + \sqrt{1-\omega^2}\right), \qquad \operatorname{acos}(\omega) = -I\log\left(\omega + I\sqrt{1-\omega^2}\right),$$

$$\operatorname{atan}(\omega) = -I\operatorname{atanh}(I\omega), \qquad \operatorname{acot}(\omega) = -I\tfrac{\log(\omega-I)-\log(\omega+I)}{2}.$$

0.4. Interactive computational sessions

When `using Grassmann` in a session, the `cayley` table can be used to recall geometric algebra information, e.g. to compare $>$ and $>>$ contractions:

```
cayley(Submanifold(3),*) # Clifford geometric product *
```

$*$	v	v_1	v_2	v_3	v_{12}	v_{13}	v_{23}	v_{123}
v	v	v_1	v_2	v_3	v_{12}	v_{13}	v_{23}	v_{123}
v_1	v_1	v	v_{12}	v_{13}	v_2	v_3	v_{123}	v_{23}
v_2	v_2	$-v_{12}$	v	v_{23}	$-v_1$	$-v_{123}$	v_3	$-v_{13}$
v_3	v_3	$-v_{13}$	$-v_{23}$	v	v_{123}	$-v_1$	$-v_2$	v_{12}
v_{12}	v_{12}	$-v_2$	v_1	v_{123}	$-v$	$-v_{23}$	v_{13}	$-v_3$
v_{13}	v_{13}	$-v_3$	$-v_{123}$	v_1	v_{23}	$-v$	$-v_{12}$	v_2
v_{23}	v_{23}	v_{123}	$-v_3$	v_2	$-v_{13}$	v_{12}	$-v$	$-v_1$
v_{123}	v_{123}	v_{23}	$-v_{13}$	v_{12}	$-v_3$	v_2	$-v_1$	$-v$

Theorem 0.47 (Linear system of equations)**.** *Let* $p_0, \dots, p_n \in \Lambda^1 V$,

$$[p_1, \dots, p_n] \vee \star \sum_{i=1}^{n} \frac{p_{1\dots(i-1)} \wedge p_0 \wedge p_{(i+1)\dots n}}{p_{1\dots n}} v_i = p_0. \tag{0.11}$$

Proof. Solve for x explicitly in matrix-vector product $[p_1, \dots, p_n] \cdot x = p_0$,

$$\begin{aligned} p_0 \wedge p_{1\dots(i-1)} \wedge p_{(i+1)\dots n} &= ([p_1, \dots, p_n] \cdot x) \wedge p_{1\dots(i-1)} \wedge p_{(i+1)\dots n} \\ &= (x_i p_i) \wedge p_{1\dots(i-1)} \wedge p_{(i+1)\dots n} \end{aligned}$$

hence $x = \sum_{i=1}^{n} \frac{p_{1\dots(i-1)} \wedge p_0 \wedge p_{(i+1)\dots n}}{p_{1\dots n}} v_i$ and the result follows. □

Remark 0.48. `Grassmann` methods for low dimensional linear systems are more numerically stable than Julia `Base.LinearAlegbra` methods and fast.

```
[1 2; 3 4]\[5,6] # inexact
@TensorOperator([1 2; 3 4])\Chain(5,6) # exact
```

$$\begin{bmatrix} -3.9999999999999987 \\ 4.499999999999999 \end{bmatrix}, \qquad \begin{bmatrix} -4 \\ 4.5 \end{bmatrix}$$

This means that using only exterior products there is an explicit solution to linear systems by allocating $\{p_{1\dots i} \wedge p_{i+1}\}_{i=0}^{n-1}$ and $\left\{p_{n-i} \wedge p_{(n-i+1)\dots n}\right\}_{i=0}^{n-1}$ and then taking exterior product permutations with p_0 also.

$$p_0 \in [p_1, \dots, p_n] \iff \forall i : p_{1\dots n} = p_{1\dots(i-1)} \wedge p_0 \wedge p_{(i+1)\dots n} \tag{0.12}$$

Since exterior products are oriented, it is sufficient to check the orientation of the hyperplanes with respect to the reference point for determining whether p_0 is a point contained in the simplex $p_{1\dots n}$. Thus, it is sufficient to check the orientation of all the same exterior products as when solving linear systems, while calculating a linear `inv` (inverse) involves only a partial application of this principle and requires also allocating a transposed dyadic result:

$$[p_1, \dots, p_n]^{-1} = \left(\sum_{i=1}^{n} \star \frac{p_{1\dots(i-1)} \wedge p_{(i+1)\dots n}}{((-1)^i)^{n-1} p_{1\dots n}} v_i \right)^T \tag{0.13}$$

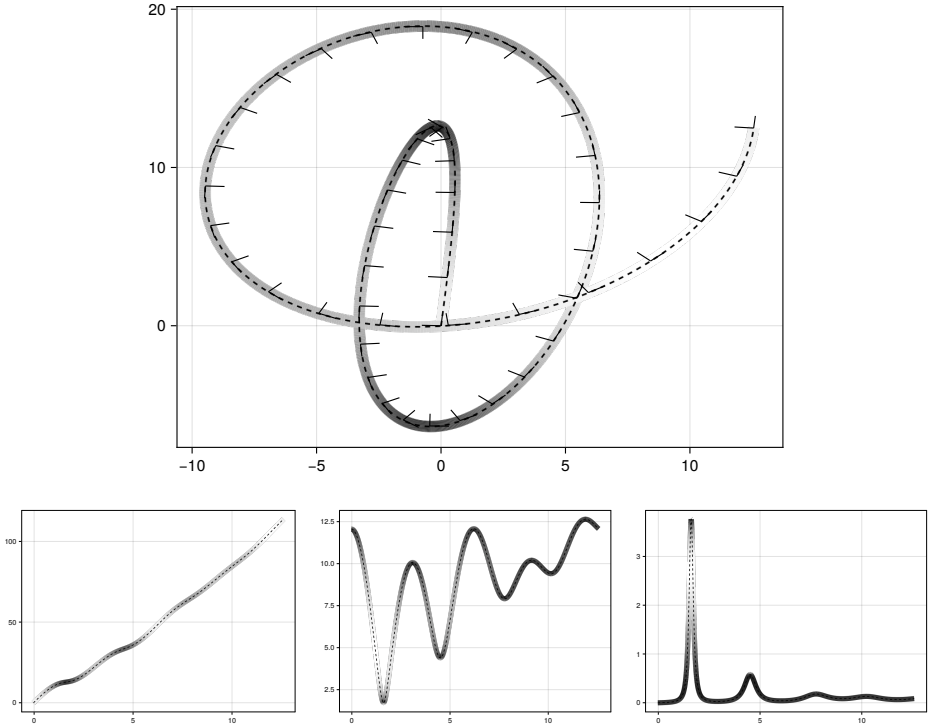
Furthermore, the $P \backslash p_0$ method implementation is a partial application of this with the action of an operator. Consider `operator` composed with `inv`

```
B = v12+2v13-3v23 # using Grassmann; basis"3"
operator(B) # convert B to endomorphisim representation
inv(operator(B))
operator(inv(B))
```

$$\begin{bmatrix} 4 & 12 & -6 \\ 12 & -6 & -4 \\ -6 & -4 & -12 \end{bmatrix}, \qquad \begin{bmatrix} 0.0204082 & 0.0612245 & -0.0306122 \\ 0.0612245 & -0.0306122 & -0.0204082 \\ -0.0306122 & -0.0204082 & -0.0612245 \end{bmatrix}$$

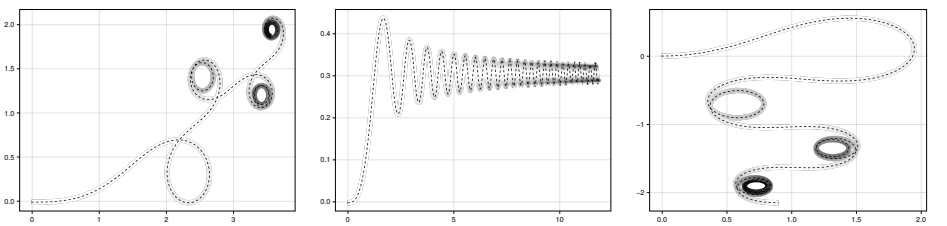
Example 0.49 (Plane curves)**.** Let t be parameter on interval from 0 to 4π

```
using Grassmann, Cartan, Makie # GLMakie
t = TensorField(0:0.01:4*pi)
lin = Chain.(cos(t)*t,sin(t)*11+t)
lines(lin); scaledarrows!(lin,unitframe(lin),gridsize=50)
lines(arclength(lin))
lines(speed(lin))
lines(curvature(lin))
```



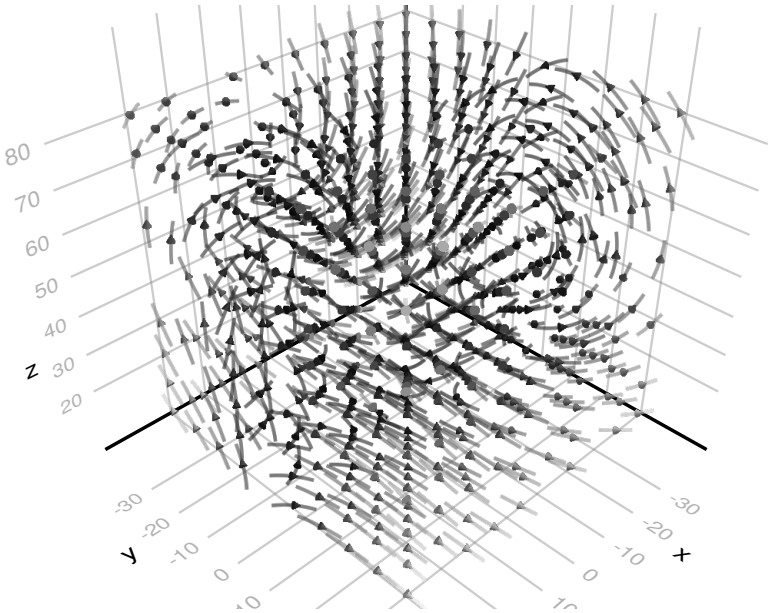
Get `curvature` from plane curve or construct `planecurve` from curvature:

```
lines(planecurve(cos(t)*t))
lines(planecurve(cos(t*t)*t))
lines(planecurve(cos(t)-t*sin(t)))
```



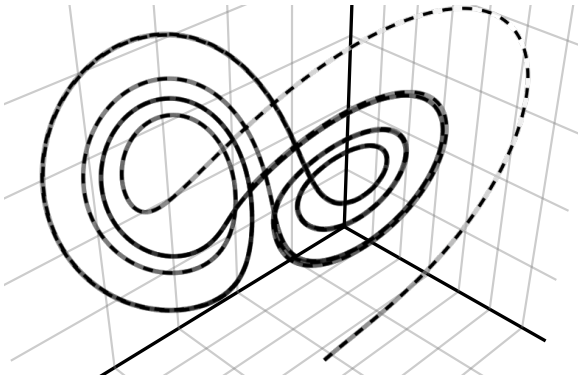
Example 0.50 (Lorenz)**.** Observe vector fields by integrating streamlines

```
Lorenz(s,r,b) = x -> Chain(
    s*(x[2]-x[1]), x[1]*(r-x[3])-x[2], x[1]*x[2]-b*x[3])
p = TensorField(ProductSpace(-40:0.2:40,-40:0.2:40,10:0.2:90))
vf = Lorenz(10.0,60.0,8/3).(p) # pick Lorenz parameters, apply
streamplot(vf,gridsize=(10,10)) # visualize vector field
```



ODE solvers in the `Adapode` package are built on `Cartan`, providing both Runge-Kutta and multistep methods with optional adaptive time stepping.

```
using Grassmann, Cartan, Adapode, Makie # GLMakie
fun,x0 = Lorenz(10.0,60.0,8/3),Chain(10.0,10.0,10.0)
ic = InitialCondition(fun,x0,2pi) # tmax = 2pi
lines(odesolve(ic,MultistepIntegrator{4}(2^-15)))
```



Recall that $\star d\omega = \star(\nabla \wedge \omega)$ and $\partial\omega = \omega > \nabla$, where $d \circ d = 0$ and $\partial \circ \partial = 0$.

Theorem 0.51 (First grade sandwich product)**.** *Reflection by hyperplane $\star\nabla$ has isometry $\omega \oslash = -\nabla \backslash \omega \nabla$.*

Proof. Let $\nabla \in \Lambda^1 V$, then $\omega = (\nabla\backslash\nabla)\omega = \nabla(d\omega + \partial\omega)$ where $\nabla \perp \partial\omega$ and $\nabla \perp d\omega$. Let's reflect across the hyperplane $\star\nabla$,

$$\begin{aligned}\nabla\backslash(d\omega - \partial\omega) &= \nabla\backslash(d\omega - \partial\omega)(\nabla\backslash\nabla) \\ &= -\nabla^2\backslash(d\omega\partial\omega)\nabla = -\partial\backslash\omega\nabla.\end{aligned}$$

Hence, reflection by hyperplane $\star\nabla$ has isometry $\omega \oslash \nabla$ which for $\nabla = v_j$ is the linear map reflecting along the j index. □

Theorem 0.52 (Cartan-Dieudonne)**.** *For every isometry of $V \to V$, there is a way to express it as composite of at most k reflections across non-singular hyperplanes. Hence there exist vectors ∇_j such that*

$$(((\omega \oslash \nabla_1) \oslash \nabla_2) \oslash \cdots) \oslash \nabla_k = \omega \oslash (\nabla_1 \nabla_2 \cdots \nabla_k) \tag{0.14}$$

for any isometry element of the orthogonal group $O(p, q)$.

Note that elements under the transformations of this group preserve inner product relations. The even grade operators make up the rotational group, where each bivector isometry is a composition of two reflections.

Exponential map and Lie group parameter special cases: consider the differential equation $\partial_i \epsilon_j = \epsilon_j \oslash \omega$, solution: $\epsilon_j(x) = \epsilon_j(0) \oslash e^{x_i \omega}$ where $\theta = 2x_i$ is Lie group paramter. Then for normalized ω,

$$e^{\theta\omega} = \sum_k \frac{(\theta\omega)^k}{k!} = \begin{cases} \cosh\theta + \omega \sinh\theta, & \text{if } \omega^2 = 1 \\ \cos\theta + \omega \sin\theta, & \text{if } \omega^2 = -1 \\ 1 + \theta\omega & \text{if } \omega^2 = 0 \end{cases}, \tag{0.15}$$

Note that $\nabla \oslash e^{\theta\omega/2} = \nabla e^{\theta\omega}$ is a double covering when using the complex numbers in the Euclidean plane.

Remark 0.53. The sandwich must be written with reversion on the left side, otherwise the rotation is clockwise and opposite of the phase parameter convention used by Euler's formula. For example, observe the resultant direction of rotation

$$e^{\frac{\pi}{4}v_{12}} v_1 \widetilde{e^{\frac{\pi}{4}v_{12}}} = -v_2$$

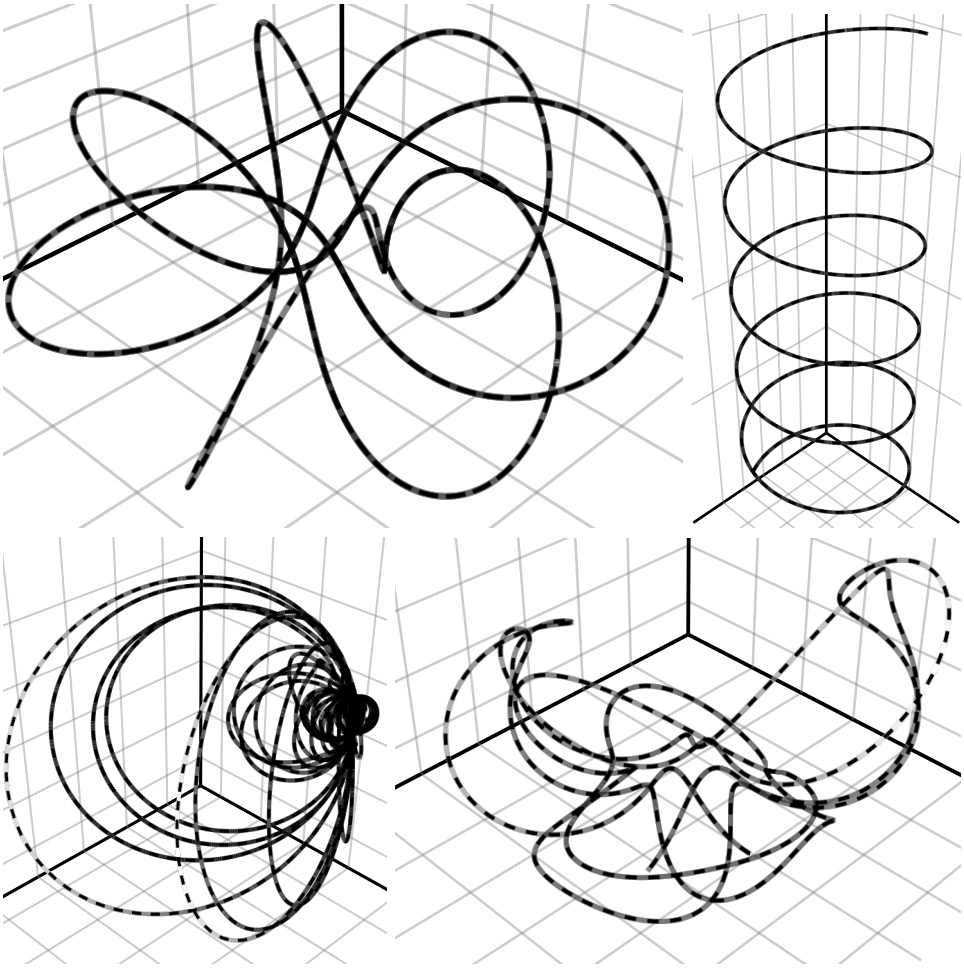
which means it is rotating in the wrong direction opposite of Euler, while

$$\widetilde{e^{\frac{\pi}{4}v_{12}}} v_1 e^{\frac{\pi}{4}v_{12}} = v_2$$

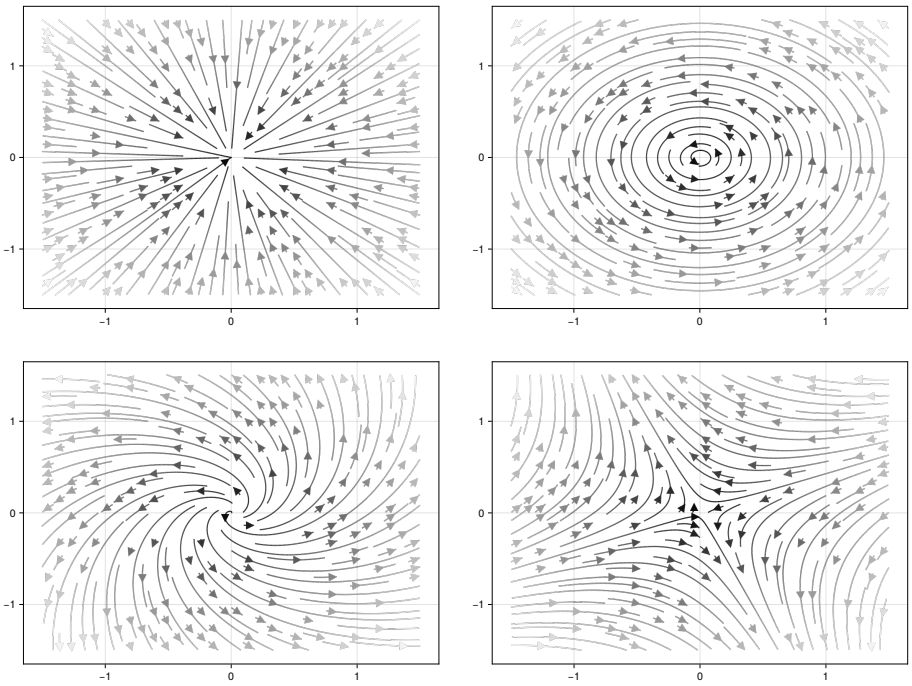
is compatible with Euler's convention. So, sandwich must be applied with its reversion on the left side–if the standard Euler rotation direction is desired. However, many authors follow the opposite convention of clockwise instead.

Example 0.54 (Riemann sphere)**.** Project $\uparrow\colon \omega \mapsto (2\omega+v_\infty(\omega^2-1))/(\omega^2+1)$ and then apply rotation before rejecting down $\downarrow\colon \omega \mapsto ((\omega\wedge b)v_\infty)/(1-v_\infty\cdot\omega)$.

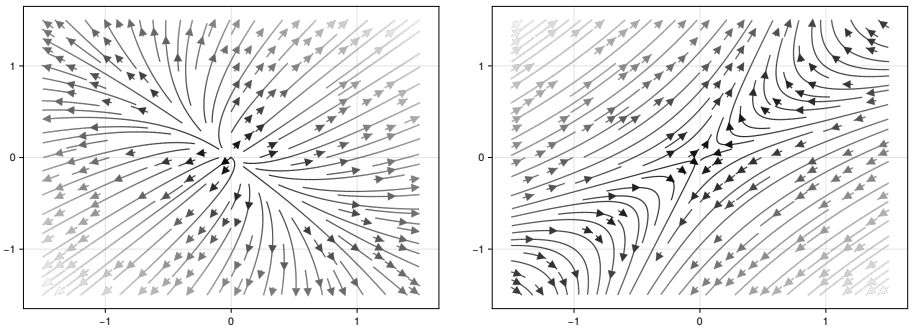
```
using Grassmann, Cartan, Makie # GLMakie
pts = TensorField(-2*pi:0.0001:2*pi)
@basis S"∞+++" # Riemann sphere
f(t) = ↓(exp(pi*t*((3/7)*v12+v∞3))>>>↑(v1+v2+v3))
f(t) = ↓(exp(t*v∞*(sin(3t)*3v1+cos(2t)*7v2-sin(5t)*4v3)/2)>>>↑(v1+v2-v3))
f(t) = ↓(exp(t*(v12+0.07v∞*(sin(3t)*3v1+cos(2t)*7v2-sin(5t)*4v3)/2))>>>↑(v1+v2-v3))
lines(V(2,3,4).(f.(pts))) # applies to each f(t)
@basis S"∞∅+++" # conformal geometric algebra
f(t) = ↓(exp(pi*t*((3/7)*v12+v∞3))>>>↑(v1+v2+v3))
lines(V(3,4,5).(f.(pts)))
```



Example 0.55 (Bivector)**.**
```
using Grassmann, Cartan, Makie #GLMakie
basis"2" # Euclidean geometric algebra in 2 dimensions
vdom = TensorField(ProductSpace{V}(-1.5:0.1:1.5,-1.5:0.1:1.5))
streamplot(tensorfield(exp(pi*v12/2)).(vdom))
streamplot(tensorfield(exp((pi/2)*v12/2)).(vdom))
streamplot(tensorfield(exp((pi/4)*v12/2)).(vdom))
streamplot(tensorfield(v1*exp((pi/4)*v12/2)).(vdom))
```

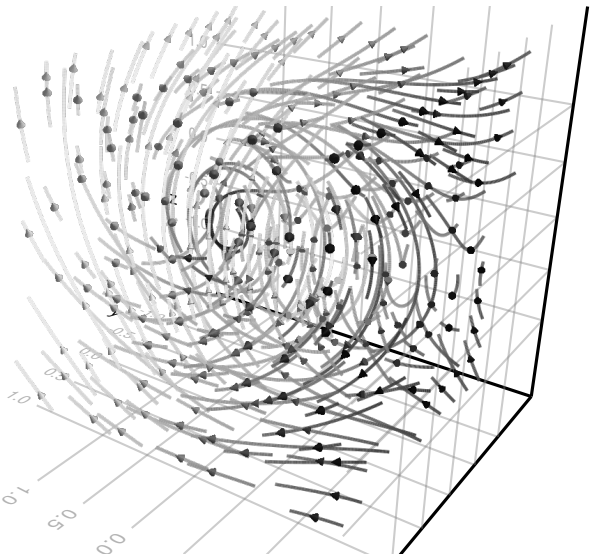


```
@basis S"+-" # Geometric algebra with Lobachevskian plane
vdom = TensorField(ProductSpace{V}(-1.5:0.1:1.5,-1.5:0.1:1.5))
streamplot(tensorfield(exp((pi/8)*v12/2)).(vdom))
streamplot(tensorfield(v1*exp((pi/4)*v12/2)).(vdom))
```

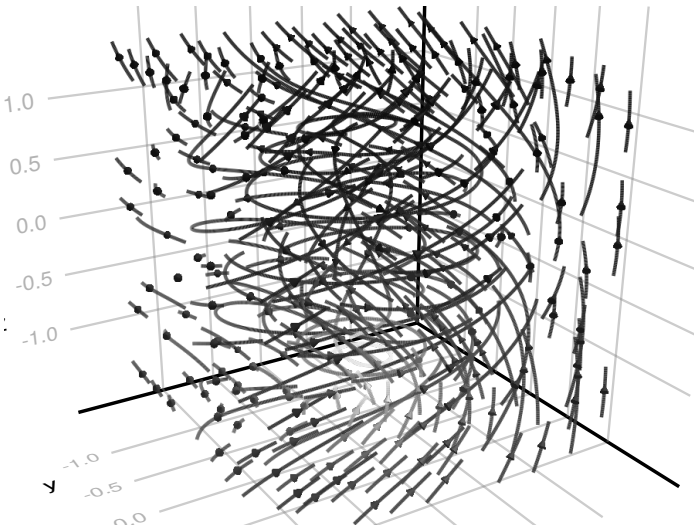


Example 0.56. `using Grassmann, Cartan, Makie; @basis S"∞+++"`

```
vdom1 = TensorField(ProductSpace{V(1,2,3)}(
    -1.5:0.1:1.5,-1.5:0.1:1.5,-1.5:0.1:1.5));
tf1 = tensorfield(exp((pi/4)*(v12+v∞3)),V(2,3,4)).(vdom1)
streamplot(tf1,gridsize=(10,10))
```



```
vdom2 = TensorField(ProductSpace{V(2,3,4)}(
    -1.5:0.1:1.5,-1.5:0.1:1.5,-1.5:0.1:1.5));
tf2 = tensorfield(exp((pi/4)*(v12+v∞3)),V(2,3,4)).(vdom2)
streamplot(tf2,gridsize=(10,10))
```



Definition 0.57. Let $[\cdot,\dots,\cdot]:\mathfrak{g}^n\to\mathfrak{g}$ define the n-linear Lie bracket with

$$[X_1,\dots,X_i,\dots,X_n]=\sum_{\sigma\in S_n}\varepsilon(\sigma)X_{\sigma(1)}(\dots(X_{\sigma(i)}(\dots(X_{\sigma(n)})\dots))\dots).$$

In `Grassmann` and `Cartan` this definition can be accessed with `Lie[Xi...]`.

In 2024, the author proved a new *multilinear Lie bracket recursion formula.*

Theorem 0.58 (Lie bracket recursion)**.** *n-bracket is sum of $(n-1)$-brackets:*

$$[X_1,\dots,X_n]=\sum_{i=1}^{n}(-1)^{i-1}X_i([X_1,\dots,X_{i-1},X_{i+1},\dots,X_n]) \tag{0.16}$$

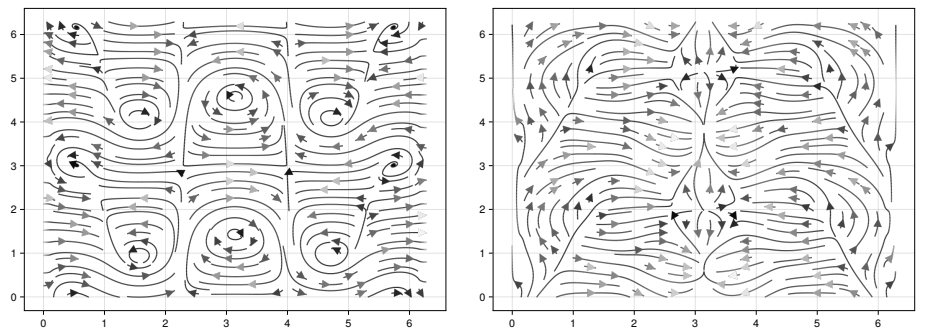
This recursion can be explicitly expanded from the unary rule $[X]=X$,

$$[X,Y]=X([Y])-Y([X]),$$
$$[X,Y,Z]=X([Y,Z])-Y([X,Z])+Z([X,Y]),$$
$$[W,X,Y,Z]=W([X,Y,Z])-X([W,Y,Z])+Y([W,X,Z])-Z([W,X,Y]),$$
$$[V,W,X,Y,Z]=V([W,X,Y,Z])-W([V,X,Y,Z])+X([V,W,Y,Z])-Y([V,W,X,Z])+Z([V,W,X,Y]).$$

The multilinear Lie bracket recursion properly generalizes the bilinear Lie bracket to the n-linear carse and is analogous to the Koszul complex of the Grassmann algebra; but is fundamentally different due to multilinear Lie bracket being non-associative, unlike the analogous exterior product.

Example 0.59 (Bracket)**.**

```
using Grassmann, Cartan, Makie # GLMakie
f1(x) = Chain(cos(3x[1]),sin(2x[1]))
f2(x) = sin(x[1]/2)*sin(x[2])
f3(x) = Chain(cos(x[1])*cos(x[2]),sin(x[2])*sin(x[1]))
vf1 = f1.(TorusParameter(100,100))
vf2 = gradient(f2.(TorusParameter(100,100)))
vf3 = f3.(TorusParameter(100,100))
lie1 = Lie[vf1,vf2] # Lie[vf1,vf2] == -Lie[vf2,vf1]
lie2 = Lie[vf1,vf2,vf3] # ternary Lie bracket
streamplot(lie1); streamplot(lie2)
```



Example 0.60 ($\int_\Omega 1$)**.** Length of line, area of disk, and volume of ball

```
linspace = ProductSpace(-2:0.03:2) # using Grassmann, Cartan
diameter = TensorField(linspace, x->abs(x)<1) # radius = 1
(integrate(diameter),2.0) # grid doesn't exactly align on 1.0
```

$$(1.98v_1, 2.0v_1)$$

```
square = ProductSpace(-2:0.003:2,-2:0.003:2)
disk = TensorField(square, x->abs(x)<1) # radius = 1
(integrate(disk),1pi)
```

$$(3.141414000000001v_{12}, 3.141592653589793v_{12})$$

```
cube = ProductSpace(-2:0.07:2,-2:0.07:2,-2:0.07:2)
ball = TensorField(cube, x->abs(x)<1) # radius = 1
(integrate(ball),4pi/3)
```

$$(4.180680595387064v_{123}, 4.1887902047863905v_{123})$$

However, this is inefficient numerical integration, for example the $58\times58\times58$ cube has $195,112$ terms and the 1334×1334 square has $1,779,556$ terms.

Theorem 0.61 (Hyper-area of hypersurface)**.** *Let $\gamma : X \subset \mathbb{R}^n \to \mathbb{R}^{n+1}$ be parameterized hypersurface $\partial(\Omega) = \gamma(X)$. Since the pullback $\gamma^*(1)$ is $\det d\gamma$,*

$$\int_{\partial(\Omega)} 1 = \int_{\gamma(X)} 1 = \int_X |\det d\gamma| = \int_X \left|\frac{\partial\gamma}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma}{\partial x_n}\right| \tag{0.17}$$

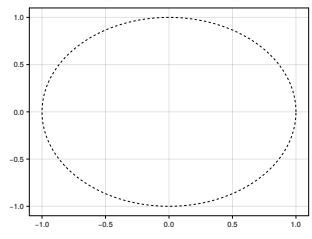
$$= \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} |\det d\gamma| = \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} \left|\frac{\partial\gamma}{\partial x_1} \wedge \cdots \wedge \frac{\partial\gamma}{\partial x_n}\right|. \tag{0.18}$$

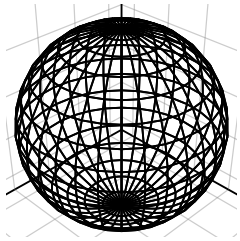
Example 0.62. Disk circumference, sphere spat `using Grassmann,Cartan`

```
t = TensorField(0:0.001:2pi)
circ = Chain.(cos(t),sin(t))
spher(x) = Chain(
    cos(x[2])*sin(x[1]), sin(x[2])*sin(x[1]),
    cos(x[1])) # GeographicParameter is swapped convention
sph = spher.(SphereParameter(60,60))
[surfacearea(circ), 2pi] # or totalarclength for AbstractCurve
[surfacearea(sph), 4pi]
lines(circ); wireframe(sph) # using Makie # GLMakie
```

$$\begin{bmatrix}6.283000000652752\\6.283185307179586\end{bmatrix}$$

$$\begin{bmatrix}12.533742943601457\\12.566370614359172\end{bmatrix}$$





Theorem 0.63 (Sector integral)**.** *Let $X \subset \mathbb{R}^n$ and $\gamma : X \to \partial(\Omega) \subset \mathbb{R}^{n+1}$ is parameterized hypersurface $\partial(\Omega) = \gamma(X)$ with $\gamma(x) = \gamma(x_1, \ldots, x_n)$, then*

$$\int_\Omega 1 = \frac{\rho^n}{n+1} \int_X \gamma \wedge \frac{\partial \gamma}{\partial x_1} \wedge \cdots \wedge \frac{\partial \gamma}{\partial x_n} \tag{0.19}$$

so there exists Ω defining the sector of hypersurface $\gamma(X)$ with scale $\rho = 1$.

Proof. Let $f : (0, \rho] \times X \to \mathbb{R}^{n+1}$ with $f(x_0, x_1, \ldots, x_n) = x_0 \cdot \gamma(x_1, \ldots, x_n)$,

$$\int_0^\rho \det df = \int_0^\rho x_0^n \cdot \gamma \wedge \frac{\partial \gamma}{\partial x_1} \wedge \cdots \wedge \frac{\partial \gamma}{\partial x_n} = \frac{\rho^n \gamma}{n+1} \wedge \frac{\partial \gamma}{\partial x_1} \wedge \cdots \wedge \frac{\partial \gamma}{\partial x_n}. \tag{0.20}$$

Since the pullback $f^*(1)$ is $\det df$, then $\Omega = f((0, \rho] \times X)$ with new variable

$$\begin{aligned} \int_\Omega 1 = \int_{f((0,\rho]\times X)} 1 = \int_X \int_0^\rho \det df = \frac{\rho^n}{n+1} \int_X \gamma \wedge \frac{\partial \gamma}{\partial x_1} \wedge \cdots \wedge \frac{d\gamma}{dx_n} \\ = \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} \int_0^\rho \det df = \frac{\rho^n}{n+1} \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} \gamma \wedge \frac{\partial \gamma}{\partial x_1} \wedge \cdots \wedge \frac{\partial \gamma}{\partial x_n}. \end{aligned}$$

Hence $\int_\Omega 1$ is `sectorintegral` covering $\gamma(X) = \partial(\Omega)$ by parameter X. □

Example 0.64 ($\int_\Omega 1$)**.** Recall `circ,sph` to evaluate $(3.1415 v_{12}, 4.17791 v_{123})$

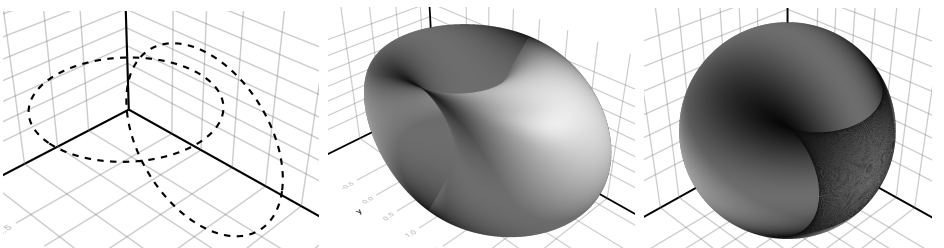
```
(sectorintegrate(circ),sectorintegrate(sph)) # more efficient
```

Example 0.65 (Link number)**.** Define the `linkmap` of two `SpaceCurve` instances $f(t)$ and $g(s)$ as parameterized hypersurface $\ell(s,t) = g(s) - f(t)$. As a corollary of the *sector integral theorem* combined with unit linkmap:

$$\frac{1}{4\pi} \int_X \gamma \wedge \frac{\partial \gamma}{\partial t} \wedge \frac{\partial \gamma}{\partial s} = \frac{1}{4\pi} \int_X \frac{g(s) - f(t)}{|g(s) - f(t)|^3} \wedge \frac{df(t)}{dt} \wedge \frac{dg(s)}{ds}, \tag{0.21}$$

therefore Gauss `linknumber` is $\frac{3}{4\pi}$ times `sectorintegrate` ∘ `unit` ∘ `linkmap` when evaluated with parameterized hypersurface $\gamma(s,t) = \ell(s,t)/|\ell(s,t)|$. So the `linknumber` divides `sectorintegral` of $\gamma(X)$ by the volume of ball Ω.

```
t = TensorField(0:0.01:2pi) # using Grassmann, Cartan, Makie
f(t) = Chain(cos(t[1]),sin(t[1]),0)
g(t) = Chain(0,1+cos(t[1]),sin(t[1]))
lines(f.(t)); lines!(g.(t)); (linknumber(f.(t),g.(t)), 1.0)
mesh(linkmap(f.(t),g.(t)),normalnorm)
mesh(unit(linkmap(f.(t),g.(t))),normalnorm)
```



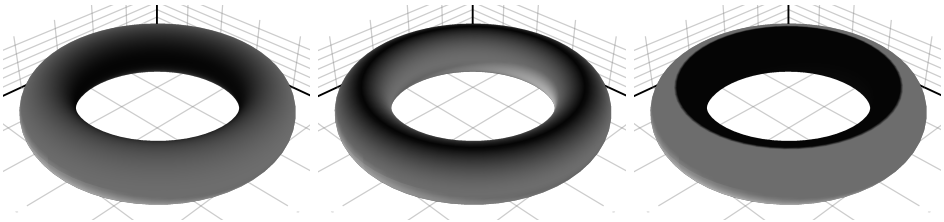
Theorem 0.66 (Gauss curvature)**.** *New alternative formulas for (extrinsic) Gauss curvature K_e and for (intrinsic) Gauss curvature K_i with normal N,*

$$K_e(x) = \nu(x) \wedge \frac{\partial \nu(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial \nu(x)}{\partial x_n}, \qquad K_i(x) = \frac{K_e(x)}{|N(x)|},$$

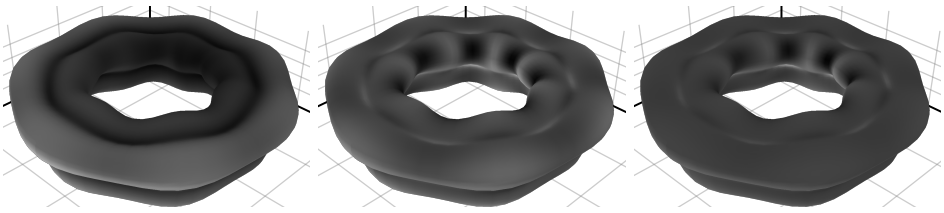
$$|K_e(x)| = \left| \star \left(\frac{\partial \nu(x)}{\partial x_1} \wedge \cdots \wedge \frac{\partial \nu(x)}{\partial x_n} \right) \right|, \qquad |K_i(x)| = \frac{|K_e(x)|}{|N(x)|}.$$

With this formula, the Gauss-Bonnet integral is a `sectorintegral` *theorem.*

Example 0.67 (Torus)**.**
```
using Grassmann, Cartan, Makie # GLMakie
torus(x) = Chain(
    (2+0.5cos(x[1]))*cos(x[2]),
    (2+0.5cos(x[1]))*sin(x[2]),
    0.5sin(x[1]))
tor = torus.(TorusParameter(60,60))
mesh(tor,normalnorm)
mesh(tor,meancurvature)
mesh(tor,gausssign)
```



Example 0.68 (Wiggle)**.**
```
using Grassmann, Cartan, Makie # GLMakie
wobble(x) = (1+0.3sin(3x[1])+0.1cos(7x[2]))
wumble(x) = (3+0.5cos(x[2]))
wiggle(x) = Chain(
    (wumble(x)+wobble(x)*cos(x[1]))*cos(x[2]),
    (wumble(x)+wobble(x)*cos(x[1]))*sin(x[2]),
    wobble(x)*sin(x[1]))
wig = wiggle.(TorusParameter(60,60))
mesh(wig,normalnorm)
mesh(wig,gaussextrinsic)
mesh(wig,gaussintrinsic)
```



Definition 0.69. When there is a Levi-Civita connection with zero torsion related to a `metrictensor`, then $\nabla_X Y - \nabla_Y X = [X, Y]$ and there exist Christoffel symbols of the `secondkind` $\Gamma_{ij}^k = \Gamma_{ji}^k$ with $\nabla_{\partial_i}\partial_j = \sum_k \Gamma_{ij}^k \partial_k$. In particular, these can be expressed in terms of the `metrictensor` g as

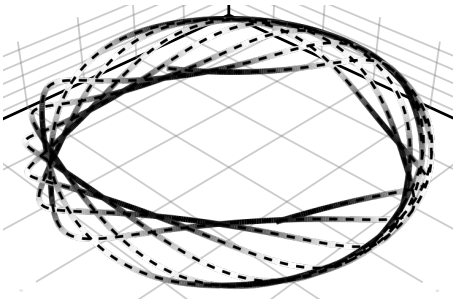
$$\Gamma_{ij}^k = \frac{1}{2}\sum_m g^{km}\left\{\frac{\partial g_{mj}}{\partial x_i} + \frac{\partial g_{im}}{\partial x_j} - \frac{\partial g_{ij}}{\partial x_m}\right\}. \tag{0.22}$$

Local geodesic differential equations for Riemannian geometry are then

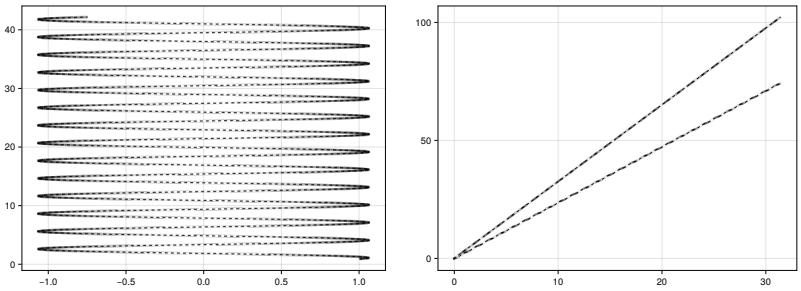
$$\frac{d^2 x_k}{dt^2} + \sum_{ij}\Gamma_{ij}^k \frac{dx_i}{dt}\frac{dx_j}{dt} = 0. \tag{0.23}$$

Example 0.70.
```
using Grassmann, Cartan, Adapode, Makie #GLMakie

tormet = surfacemetric(tor) # intrinsic metric
torcoef = secondkind(tormet) # Christoffel symbols
ic = geodesic(torcoef,Chain(1.0,1.0),Chain(1.0,sqrt(2)),10pi)
sol = geosolve(ic,ExplicitIntegrator{4}(2^-7)) # Runge-Kutta
lines(torus.(sol))
```

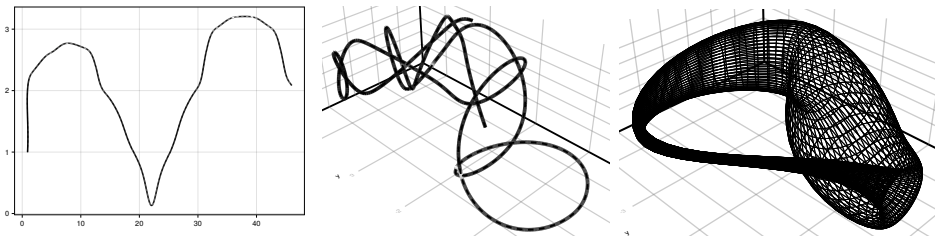


```
totalarclength(sol) # apparent length of parameter path
@basis MetricTensor([1 1; 1 1]) # abstract non-Euclidean V
solm = TensorField(tormet(sol),Chain{V}.(value.(fiber(sol))))
totalarclength(solm) # 2D estimate totalarclength(klein.(sol))
totalarclength(klein.(sol)) # compute in 3D Euclidean metric
lines(solm) # parametric curve can have non-Euclidean metric
lines(arclength(solm)); lines!(arclength(sol))
```



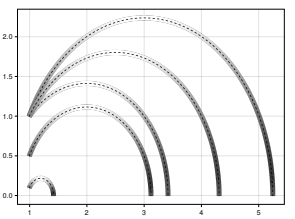
Example 0.71 (Klein geodesic)**.** General `ImmersedTopology` are supported

```
klein(x) = klein(x[1],x[2]/2)
function klein(v,u)
    x = cos(u)*(-2/15)*(3cos(v)-30sin(u)+90sin(u)*cos(u)^4-
        60sin(u)*cos(u)^6+5cos(u)*cos(v)*sin(u))
    y = sin(u)*(-1/15)*(3cos(v)-3cos(v)*cos(u)^2-
        48cos(v)*cos(u)^4+48cos(v)*cos(u)^6-
        60sin(u)+5cos(u)*cos(v)*sin(u)-
        5cos(v)*sin(u)*cos(u)^3-80cos(v)*sin(u)*cos(u)^5+
        80cos(v)*sin(u)*cos(u)^7)
    z = sin(v)*(2/15)*(3+5cos(u)*sin(u))
    Chain(x,y,z)
end # too long paths over QuotientTopology can stack overflow
kle = klein.(KleinParameter(100,100))
klecoef = secondkind(surfacemetric(kle))
ic = geodesic(klecoef,Chain(1.0,1.0),Chain(1.0,2.0),2pi)
lines(geosolve(ic,ExplicitIntegrator{4}(2^-7)));wireframe(kle)
```



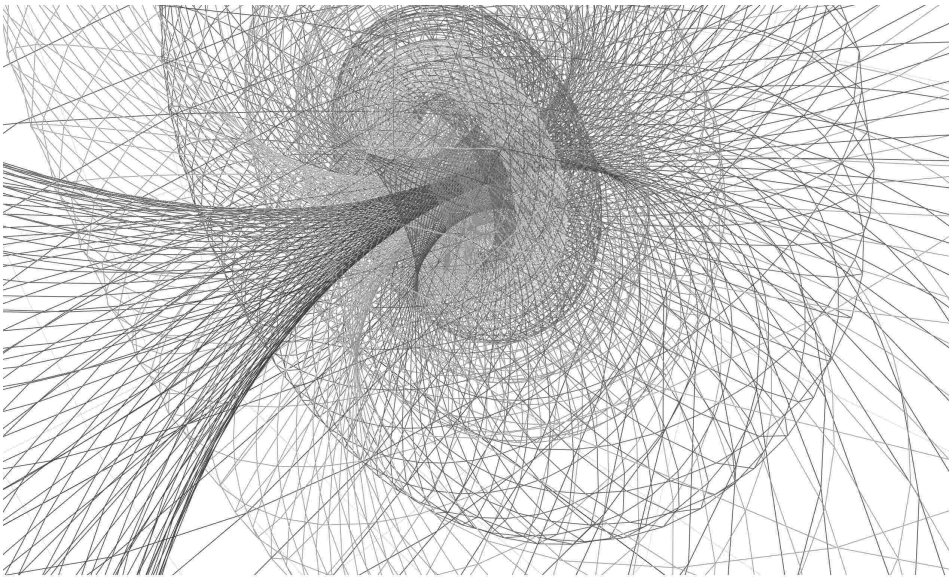
Example 0.72 (Upper half plane)**.** Intrinsic hyperbolic Lobachevsky metric

```
halfplane(x) = TensorOperator(Chain(
    Chain(Chain(0.0,inv(x[2])),Chain(-inv(x[2]),0.0)),
    Chain(Chain(-inv(x[2]),0.0),Chain(0.0,-inv(x[2])))))
z1 = geosolve(halfplane,Chain(1.0,1.0),Chain(1.0,2.0),10pi,7)
z2 = geosolve(halfplane,Chain(1.0,0.1),Chain(1.0,2.0),10pi,7)
z3 = geosolve(halfplane,Chain(1.0,0.5),Chain(1.0,2.0),10pi,7)
z4 = geosolve(halfplane,Chain(1.0,1.0),Chain(1.0,1.0),10pi,7)
z5 = geosolve(halfplane,Chain(1.0,1.0),Chain(1.0,1.5),10pi,7)
lines(z1); lines!(z2); lines!(z3); lines!(z4); lines!(z5)
```



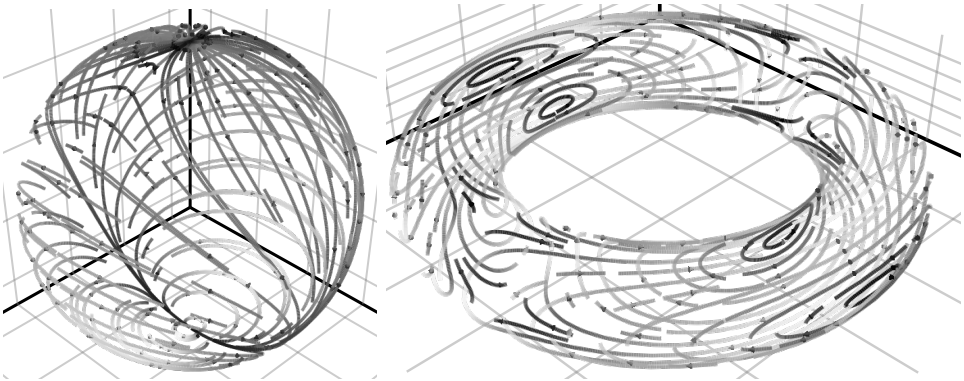
Example 0.73. Calculus over Hopf fibration is enabled by `HopfTopology`,

```
stereohopf(x) = stereohopf(x[1],x[2],x[3])
function stereohopf(theta,phi,psi)
    a = cos(theta)*exp((im/2)*(psi-phi))
    b = sin(theta)*exp((im/2)*(psi+phi))
    Chain(imag(a),real(b),imag(b))/(1-real(a))
end
hs = stereohopf.(HopfParameter());
alteration!(hs,wireframe,wireframe!)
```



Example 0.74. Streamplots on tangent spaces enabled by `Cartan` methods,

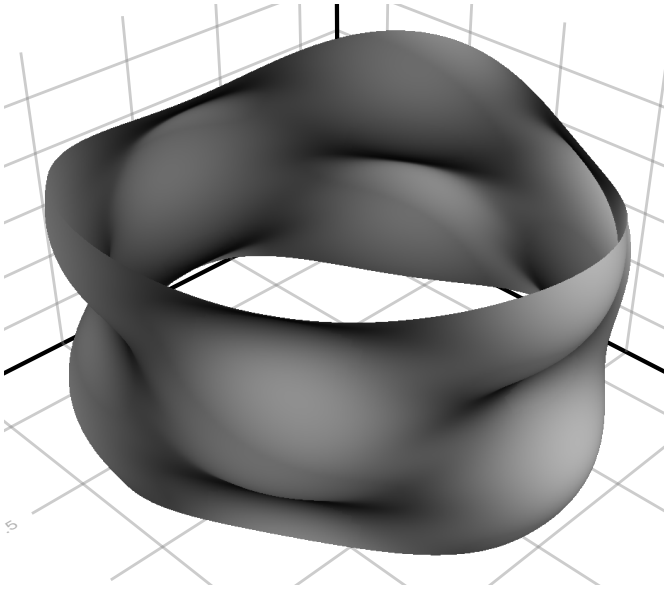
```
streamplot(sph,vf2) # recall from previous examples
streamplot(tor,vf3)
```



Example 0.75 (da Rios)**.** The `Cartan` abstractions enable easily integrating

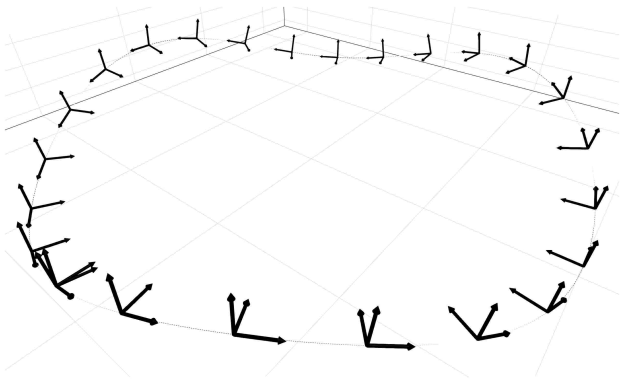
$$\frac{\partial \gamma(x)}{\partial x_2} = \star\left(\frac{\partial \gamma(x)}{\partial x_1} \wedge \frac{\partial^2 \gamma(x)}{\partial x_1^2}\right) \tag{0.24}$$

```
using Grassmann, Cartan, Adapode, Makie # GLMakie
start(x) = Chain(cos(x),sin(x),cos(1.5x)*sin(1.5x)/5)
x1 = start.(TorusParameter(180));
darios(t,dt=tangent(fiber(t))) = hodge(wedge(dt,tangent(dt)))
sol = odesolve(darios,x1,1.0,2^-11); mesh(sol,normalnorm)
```



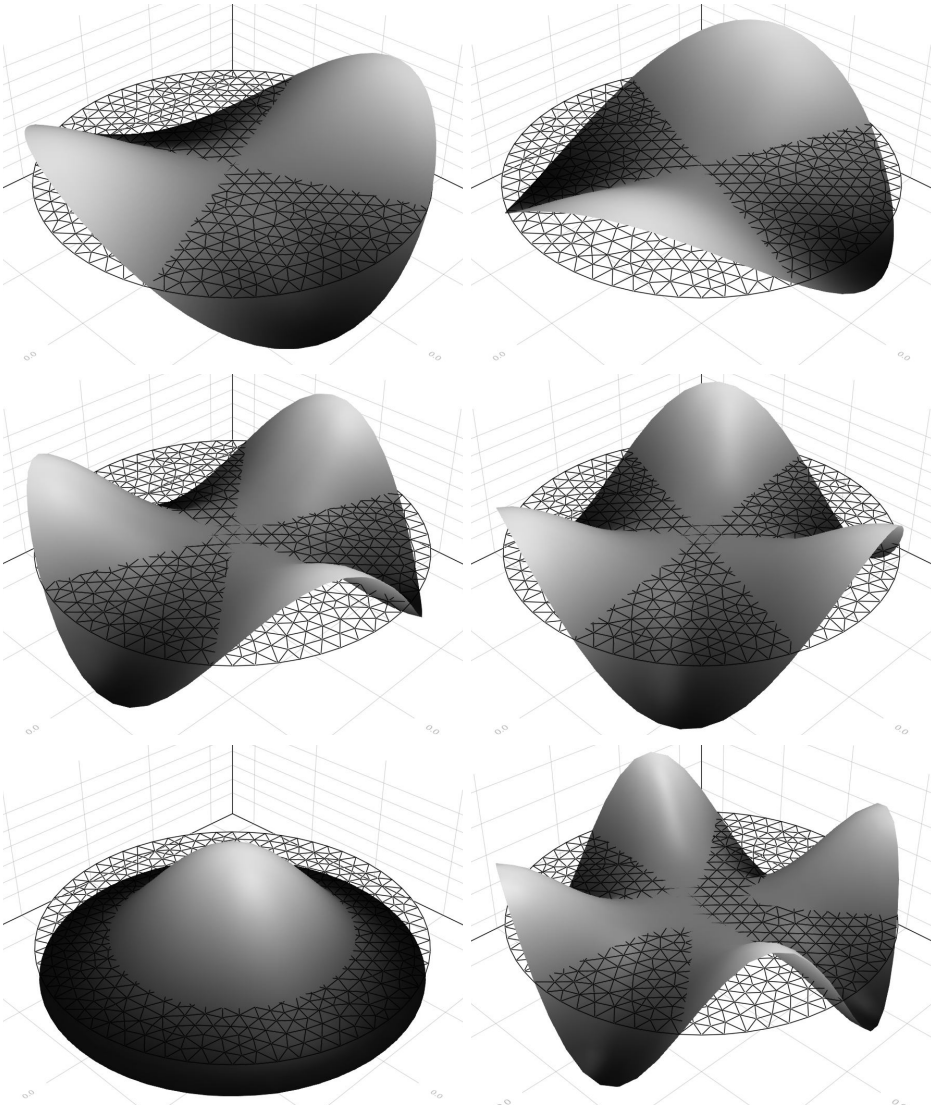
Example 0.76 (Bishop frame)**.** As an alternative to the standard Frenet style `unitframe`, the `bishopunitframe(::SpaceCurve,angle::Real)` has an optional angle (integration constant) modulo rotation of tangent axis.

```
scaledarrows(x1,bishopunitframe(x1),gridsize=25)
lines!(x1,linestyle=:dash) # angle is optional
```



Example 0.77 (Eigenmodes of disk)**.** $-\Delta u = \lambda u$ with boundary $n \cdot \nabla u = 0$ is enabled with `assemble` for stiffness and mass matrix from `Adapode`:

```
using Grassmann, Cartan, Adapode, MATLAB, Makie # GLMakie
pt,pe = initmesh("circleg","hmax"=>0.1) # MATLAB circleg mesh
A,M = assemble(pt,1,1,0) # stiffness & mass matrix assembly
using KrylovKit # provides general eigsolve
yi,xi = geneigsolve((A,M),10,:SR;krylovdim=100) # maxiter=100
amp = TensorField.(Ref(pt),xi./3) # solutions amplitude
mode = TensorField.(graphbundle.(amp),xi) # make 3D surface
mesh(mode[7]); wireframe!(pt) # figure modes are 4,5,7,8,6,9
```



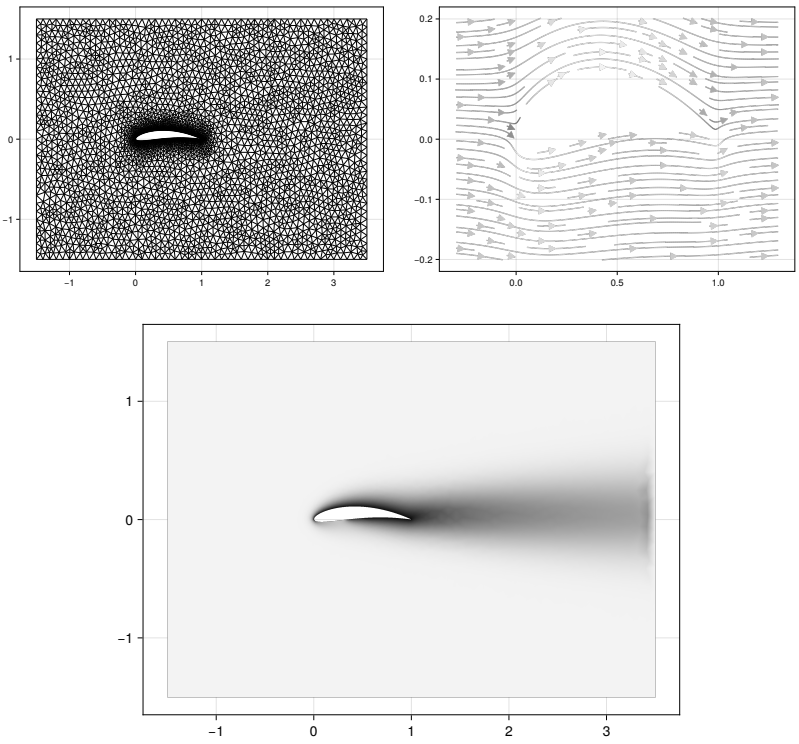
To build on the `FiberBundle` functionality of `Cartan`, the numerical analysis package `Adapode` is being developed to provide extra utilities for finite element method assemblies. Poisson equation ($-\nabla\cdot(c\nabla u) = f$) syntax or transport ($-\epsilon\nabla^2 u + c\cdot\nabla u = f$) equations with finite element methods can be expressed in terms of methods like `volumes` using exterior products or `gradienthat` by applying the exterior algebra principles discussed. Global `Grassmann` element assembly problems involve applying geometric algebra locally per element basis and combining it with a global manifold topology.

```
function solvepoisson(t,e,c,f,k,gD=0,gN=0)
    m = volumes(t)
    b = assembleload(t,f,m)
    A = assemblestiffness(t,c,m)
    R,r = assemblerobin(e,k,gD,gN)
    return TensorField(t,(A+R)\(b+r))
end
function solvetransport(t,e,c,f=1,eps=0.1)
    m = volumes(t)
    g = gradienthat(t,m)
    A = assemblestiffness(t,eps,m,g)
    b = assembleload(t,f,m)
    C = assembleconvection(t,c,m,g)
    TensorField(t,solvedirichlet(A+C,b,e))
end
function solvetransportdiffusion(tf,ek,c,d,gD=0,gN=0)
    t,f,e,k = base(tf),fiber(tf),base(ek),fiber(ek)
    m = volumes(t)
    g = gradienthat(t,m)
    A = assemblestiffness(t,c,m,g)
    b = means(immersion(t),f)
    C = assembleconvection(t,b,m,g)
    Sd = assembleSD(t,sqrt(d)*b,m,g)
    R,r = assemblerobin(e,k,gD,gN)
    return TensorField(t,(A+R-C'+Sd)\r)
end
```

More general problems for finite element boundary value problems are also enabled by mesh representations imported into `Cartan` from external sources and computationally operated on in terms of `Grassmann` algebra. Many of these methods automatically generalize to higher dimensional manifolds and are compatible with discrete differential geometry. Further advanced features such as `DiscontinuousTopology` have been implemented and the `LagrangeTopology` variant of `SimplexTopology` is being used in research.

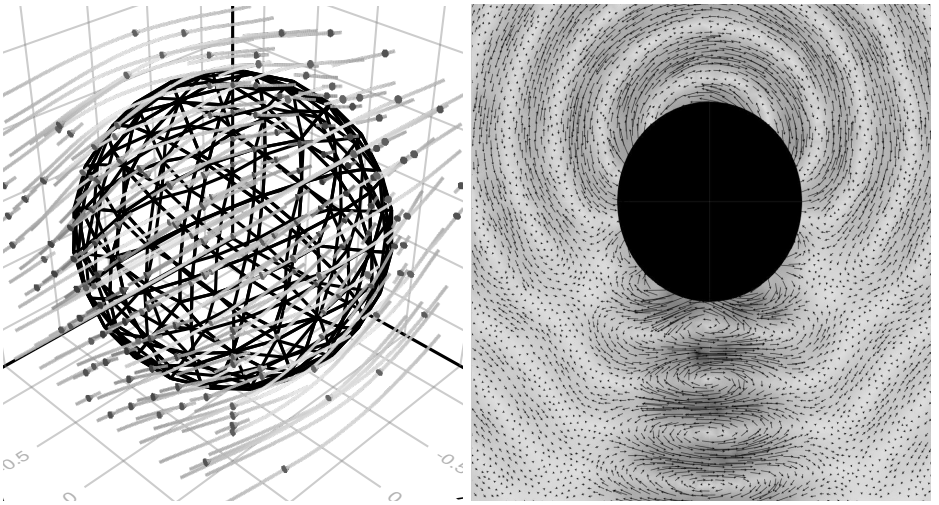
Example 0.78 (Heatflow around airfoil)**.** `FlowGeometry` builds on `Cartan` to provide NACA airfoil shapes, and `Adapode` can solve transport diffusion.

```
using Grassmann, Cartan, Adapode, FlowGeometry, MATLAB, Makie
pt,pe = initmesh(decsg(NACA"6511"),"hmax"=>0.1)
tf = solvepoisson(pt,pe,1,0,
    x->(x[2]>3.49 ? 1e6 : 0.0),0,x->(x[2]<-1.49 ? 1.0 : 0.0))
function kappa(z); x = base(z)
    if x[2]<-1.49 || sqrt((x[2]-0.5)^2+x[3]^2)<0.51
        1e6
    else
        x[2]>3.49 ? fiber(z)[1] : 0.0
    end
end
gtf = -gradient(tf)
kf = kappa.(gtf(immersion(pe)))
tf2 = solvetransportdiffusion(gtf,kf,0.01,1/50,
    x->(sqrt((x[2]-0.5)^2+x[3]^2)<0.7 ? 1.0 : 0.0))
wireframe(pt)
streamplot(gtf,-0.3..1.3,-0.2..0.2)
mesh(tf2)
```



Example 0.79. Most finite element methods `using Grassmann, Cartan` automatically generalize to higher dimension manifolds with e.g. tetrahedra, and the author has contributed to packages such as *Triangulate.jl*, *TetGen.jl*.

```
using Grassmann, Cartan, Adapode,FlowGeometry,MiniQhull,TetGen
ps = sphere(sphere(∂(delaunay(PointCloud(sphere())))))
pt,pe = tetrahedralize(cubesphere(),"vpq1.414a0.1";
    holes=[TetGen.Point(0.0,0.0,0.0)])
tf = solvepoisson(pt,pe,1,0,
    x->(x[2]>1.99 ? 1e6 : 0.0),0,x->(x[2]<-1.99 ? 1.0 : 0.0))
streamplot(-gradient(tf),-1.1..1.1,-1.1..1.1,-1.1..1.1,
    gridsize=(10,10,10))
wireframe!(ps)
```



Example 0.80 (Maxwell's equations rewritten)**.** `Cartan` has Nedelec edge interpolation useful for solving the time harmonic wave equation. Form Maxwell's equations using the Faraday bivector dA with $ddA = 0$,

$$Ev_t + \star(Bv_t) = (\nabla V - \partial_t A)v_t + \star((\star dA)v_t) = dA, \tag{0.25}$$

where E is electric field, B magnetic field, A is vector potential.

$$ddA = 0 \Longleftrightarrow \begin{cases} \partial B = 0 & \text{Gauss's law} \\ \star dE = -\partial_t B & \text{Faraday's law} \end{cases} \tag{0.26}$$

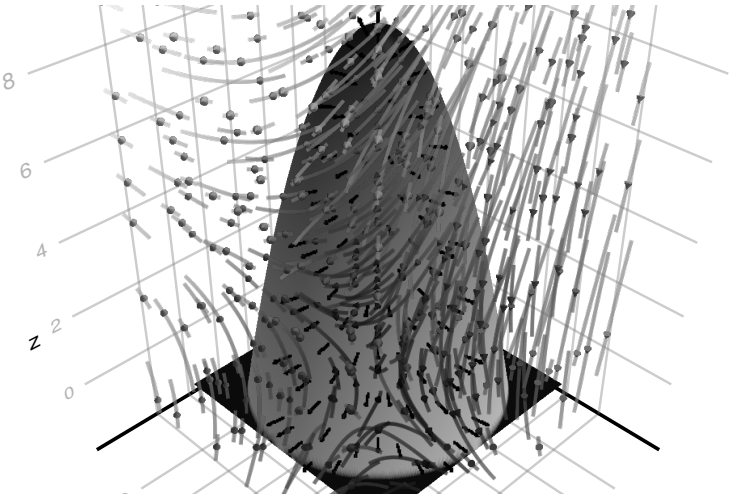
$$\star d \star dA = J \Longleftrightarrow \begin{cases} \partial E = \rho & \text{Gauss's law} \\ \star dB = J + \partial_t E & \text{Ampere's law} \end{cases} \tag{0.27}$$

Maxwell's equations simplify to a single spacetime wave equation.

$$\nabla(Ev_t + \star(Bv_t)) = \nabla dA = \star d \star dA = \nabla^2 A = J \tag{0.28}$$

Example 0.81 (Stokes theorem)**.** Paraboloid $S = \gamma(X)$ bound by compact support `disk` of radius 3, with circle $f([0, 2\pi]) = \partial(S)$, and vector field F:

```
using Grassmann, Cartan, Makie # GLMakie
square = TensorField(ProductSpace(-3:0.003:3,-3:0.003:3))
cube = TensorField(ProductSpace(-4:0.1:4,-4:0.1:4,-1:0.2:10))
disk = (x->float(abs(x)<3)).(square) # compact support
paraboloid(x) = 9-x[1]*x[1]-x[2]*x[2]
S = graph(disk*paraboloid.(square))
F(x) = Chain(2x[3]-x[2],x[1]+x[3],3x[1]-2x[2])
mesh(S,normalnorm)
scaledarrows!(S,disk*unitnormal(S),gridsize=(22,22))
streamplot!(F.(cube),gridsize=(11,11,11))
```



$$\int_S \nabla \times F \cdot dS = \int_X (\nabla \times F) \cdot \left(\frac{\partial \gamma(x)}{\partial x_1} \times \frac{\partial \gamma(x)}{\partial x_2} \right) dx_1 dx_2 \tag{0.29}$$

```
integrate(disk*(curl(F.(cube)).(S) ⋅ normal(S)))
```

$$\int_{\partial(S)} F \cdot ds = \int_0^{2\pi} F(f(t)) \cdot f'(t) dt \tag{0.30}$$

```
t = TensorField(0:0.001:2pi)
f(t) = Chain(3cos(t[1]),3sin(t[1]),0.0)

integrate(F.(f.(t)) ⋅ tangent(f.(t)))
```

$$\int_S \nabla \times F \cdot dS = \int_{\partial(S)} F \cdot ds \tag{0.31}$$

$$56.5474 \approx 56.547 \approx 56.548667764616276 \approx 18\pi$$

Both integration techniques come out to the same answer, this is called Stokes theorem, a special case of the more general Stokes-Cartan theorem.

Grassmann.jl and *Cartan.jl* pioneered many computational language design aspects for fully generalized high performance computing with differential geometric algebra. All of the mathematical types and operations in this program were implemented from scratch with fundamental mathematical principles merged to Julia's type polymorphism code generation, which has been refined and is being optimized for scientific computing over time.

This leads to the capability for multiple dispatch polymorphisms with type aliases such as `Scalar`, `GradedVector`, `Bivector`, `Trivector`, or also `Quaternion`, `Simplex`, etc. There are aliases such as `Positions`, `Interval`, `IntervalRange`, `Rectangle`, `Hyperrectangle`, `RealRegion`, `RealSpace`, or the many aliases of the type `TensorField`, such as `ElementMap`, `SimplexMap`, `FaceMap`, `IntervalMap`, `RectangleMap`, `HyperrectangleMap`, `Variation`, `ParametricMap`, `RealFunction`, `PlaneCurve`, `SpaceCurve`, `AbstractCurve`, `SurfaceGrid`, `VolumeGrid`, `ScalarGrid`, `CliffordField`, `DiagonalField`, `EndomorphismField`, `OutermorphismField`, `ComplexMap`, `PhasorField`, or `QuaternionField`, `SpinorField`, `GradedField`, `ScalarField` `VectorField` `BivectorField`, `TrivectorField`. Versatility of the `Grassmann` and `Cartan` type system opens up many possibilities for computational language design.

This is a new paradigm of geometric algebra, anti-symmetric tensor products, rotational algebras, bivector groups, and multilinear Lie brackets. Algebra based on Leibniz differentials and Grassmann's exterior calculus extended with `TensorField` sections over a `FrameBundle` yields differential geometric algebra based on the `ImmersedTopology` of a `FiberBundle`. The *sector integral theorem* is a new alternative specialization to the Stokes-Cartan theorem for general integrals in differential geometry, relating an integral on a manifold and an integral on its boundary. Sector integral theory is a new alternative formalism enabling `Cartan` style calculations.

As a result of Grassmann's exterior and interior products, the Hodge-DeRahm chain complex from cohomology theory is

$$0 \underset{\partial}{\overset{d}{\rightleftarrows}} \Omega^0(M) \underset{\partial}{\overset{d}{\rightleftarrows}} \Omega^1(M) \underset{\partial}{\overset{d}{\rightleftarrows}} \cdots \underset{\partial}{\overset{d}{\rightleftarrows}} \Omega^n(M) \underset{\partial}{\overset{d}{\rightleftarrows}} 0,$$

having dimensional equivalence brought by Grassmann-Hodge complement,

$$\mathcal{H}^{n-p}M \cong \tfrac{\ker(d\Omega^{n-p}M)}{\mathrm{im}(d\Omega^{n-p+1}M)}, \qquad \dim \mathcal{H}^p M = \dim \tfrac{\ker(\partial\Omega^p M)}{\mathrm{im}(\partial\Omega^{p+1}M)} = b_p(M).$$

The rank of the grade p boundary operator is

$$\operatorname{rank} \langle \partial\langle M\rangle_{p+1}\rangle_p = \min\left\{\dim \langle \partial\langle M\rangle_{p+1}\rangle_p, \dim\langle M\rangle_{p+1}\right\}. \tag{0.32}$$

Invariant topological information can be computed using ranks of homology

$$b_p(M) = \dim\langle M\rangle_{p+1} - \operatorname{rank} \langle \partial\langle M\rangle_{p+1}\rangle_p - \operatorname{rank} \langle \partial\langle M\rangle_{p+2}\rangle_{p+1} \tag{0.33}$$

are Betti numbers with Euler characteristic $\chi(M) = \sum_p (-1)^p b_p$.

Grassmann algebra is a unifying mathematical foundation. Improving efficiency of multi-disciplinary research using differential geometric algebra by relying on universal mathematical principles is possible. Transforming superficial knowledge into deeper understanding is then achieved with the unified foundations widely applicable to the many different sub-disciplines related to geometry and mathematical physics. During the early stages when *Grassmann.jl* and *Cartan.jl* were being developed, many new computational language design principles were pioneered for differential geometric algebra research and development with a modern interactive scientific programming language. With the interest surrounding the project increasing, there have been some other similar projects taking inspiration from the *Grassmann.jl* computational language design and thus validating the concepts.

While some of the computational language designs in *Grassmann.jl* and *Cartan.jl* may seem like obvious choices for people seeing the completed idea, please be aware that it has taken an enormous amount of creativity and effort to make the many different decisions for these projects. The style of computational language the author wanted to use didn't exist yet before, so if it really was such an obvious design–then why didn't it exist before? It took a lot of deep thinking and trying out previously overlooked ideas.

Acknowledgements: there was absolutely zero support from academic institutions for the entire *Grassmann.jl* and *Cartan.jl* development and the research project, everything was done entirely alone without any assistance. Wolfram Research employed the author for some time, but in time it turned out that the Wolfram assigned manager was delaying the author's progress in computational language design research. Since then, it appears that many researchers are using company resources (from funds paid by academia) to stalk the author with goals to gather language design inspiration. It seems to be a regular occurence that many academic "communities" typically act in a way that results in taking inspiration from the author's research while using their bureaucracy to delay the author's progress. While this author was open to implementing this research program at their institutions, the goals of these institutions was always to delay the author's progress. Typical academic institutions everywhere thrive by creating obstacles for people on a regular basis. Overall, the author's experience is that the academic "community" tend to leverage their institutional positions to delay researchers while they figure out how to benefit. Academics operate with bureaucratic levers to manipulate and delay the progress of other people or to brutally cut them off from society. Finishing this research project required avoiding academia.

The academic "community" and Julia Computing have gone out of their way to put obstacles in the way of the *Grassmann.jl* and *Cartan.jl* project.

Thanks to family and people who showed support engaging with project.

Michael Reed, *Differential geometric algebra with Leibniz and Grassmann*, JuliaCon (2019)

Michael Reed, *Foundations of differential geometric algebra* (2021)

Michael Reed, *Multilinear Lie bracket recursion formula* (2024)

Emil Artin, *Geometric Algebra* (1957)

John Browne, *Grassmann Algebra, Volume 1: Foundations* (2011)

C. Doran, D. Hestenes, F. Sommen, and N. Van Acker, *Lie groups as spin groups*, J. Math Phys. (1993)

Lachlan Gunn, Derek Abbott, James Chappell, Ashar Iqbal, *Functions of multivector variables* (2011)

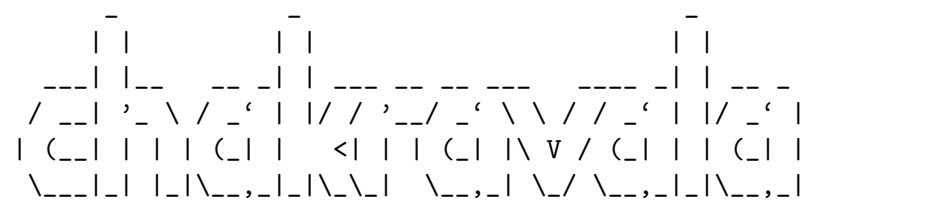
Vladimir and Tijana Ivancevic, *Undergraduate lecture notes in DeRahm-Hodge theory.* arXiv (2011)

Eckhard Hitzer, *Introduction to Clifford's Geometric Algebra* (2011)

David Hestenes, *Tutorial on geometric calculus.* AACA (2013)

Garret Sobczyk, *New Foundations in Mathematics: The Geometric Concept of Number*, Birkhauser (2013)

Siavash Shahshahani, *An Introductory Course on Differentiable Manifolds*, Dover (2016)



https://github.com/chakravala
https://crucialflow.com