Geomstats

A deep dive into discrete surfaces implementation

Luís F. Pereira

Geometric Intelligence Lab @ UCSB

Infinite-dimensional Geometry: Theory and Applications @ ESI
February 12, 2025

Acknowledgments

- based on (Hartman, 2023)[1]
- original code: emmanuel-hartman/H2_SurfaceMatch
- initial implementation by Emmanuel: see geomstats/#1711

[1] Hartman et al., "Elastic Shape Analysis of Surfaces with Second-Order Sobolev Metrics", 2023.

Setup

https://github.com/luisfpereira/esi25

Geomstats[1]

Computations and **statistics** on **manifolds** with geometric structures

- `Python` (powered by `numpy`, `autograd`, or `pytorch`)
- open-source (MIT license)
- object-oriented
- `sklearn`-inspired API
- thoroughly tested with (extended) `pytest`

[1] Guigui, Miolane, and Pennec, "Introduction to Riemannian Geometry and Geometric Statistics", 2022.

Motivation: Riemannian geometry

Let $A, B \in \mathcal{X}$, where $\mathcal{X}$ is some non-linear space:

1. how to compute **distances** between A and B?
2. how to **interpolate** between A and B?
3. how to **extrapolate**?

Why these questions?

1. notion of distance $\implies$ can do statistics
2. interpolation $\implies$ can meaningfully move between points
3. extrapolation $\implies$ strong predicting ability

Motivation: why Geomstats?

A platform implementing **consistently** and **flexibly** different Riemannian manifolds (and more).

Consequences:

- code reuse
- (hopefully) less bugs (easier testing)
- easy experimentation
- productivity gains (after mastering structure)
- sense of community

Backend

```
import geomstats.backend as gs
```

	numpy	autograd	pytorch
numerical precision		float64	
gpu			✓
automatic differentiation		✓	✓

```
import os

os.environ["GEOMSTATS_BACKEND"] = "pytorch"
```

or

```
export GEOMSTATS_BACKEND=pytorch
```

About today's presentation

- assumes familiarity with `geomstats`
- transform abstract/high-level knowledge in actual practical knowledge
- representative, but `geomstats` is much more

Every time you see
`space` or `space.metric`
think I could have instantiated your favorite manifold/metric instead.

About today's presentation

- assumes familiarity with `geomstats`
- transform abstract/high-level knowledge in actual practical knowledge
- representative, but `geomstats` is much more

Every time you see
`space` or `space.metric`
think I could have instantiated your favorite manifold/metric instead.

Dive in warning

Main goal:

introduce **parameterized surfaces**

- show **space/metric** in `geomstats`
- reveal auxiliary, but invaluable, **computational objects**
- present strategy to solve **geodesic boundary value problem**

Triangle mesh

Triangle mesh M:

1. ordered pair $(V(M), F(M))$
2. set $V(M) \subseteq \mathbb{R}^3$ of vertices
3. set $F \subseteq \{1, \ldots, |V|\}^3$ of triangular faces
4. $f = (f_1, f_2, f_3) \in F$ defines a triangular face enclosed by the corresponding vertices v_{f_1}, v_{f_2}, and v_{f_3}

Faces implicitly define the edges $E(F)$ between the vertices.

Parameterized surfaces and Sobolev metric

Set of triangle meshes with **fixed combinatorial structure** $\mathfrak{M}$.

- fixed number of vertices V
- fixed connectivities: fixed set of faces F (and edges E)

A surface is fully defined by the **location of the vertices**

Second-order Sobolev metric (aka elastic metric):

$$G_q(h,k) = \int_M \left(\langle h,k \rangle + g_q^{-1}(dh,dk) + \left\langle \Delta_q h, \Delta_q h \right\rangle \right) \mathrm{vol}_q$$

Parameterized surfaces and Sobolev metric

Set of triangle meshes with **fixed combinatorial structure** $\mathfrak{M}$.

- fixed number of vertices V
- fixed connectivities: fixed set of faces F (and edges E)

A surface is fully defined by the **location of the vertices**

Second-order Sobolev metric (aka elastic metric):

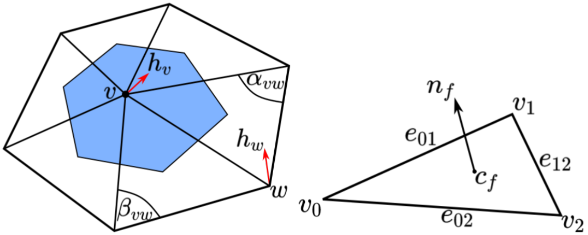
$$G_q(h,k) = \int_M \Big(\langle h, k \rangle + g_q^{-1}(dh, dk) + \big\langle \Delta_q h, \Delta_q h \big\rangle \Big) \operatorname{vol}_q$$

Discretization of (spatial) quantities[1]

- tangent vectors discretized on the vertices V:

$$h := \{h_v \in \mathbb{R}^3 \mid v \in V\} \in \mathbb{R}^{3n}$$

- first-order terms discretized on the faces F
- Laplace operator discretized on the dual cell
- volume form both at a vertex and at a face (from face to vertex)

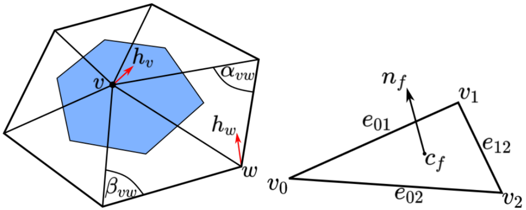


[1] Crane, "Discrete differential geometry: An applied introduction", 2018.

Discretization of (spatial) quantities (cont'd)

$$G_q(h,k) = \int_M \left(\langle h, k \rangle + g_q^{-1}(dh, dk) + \langle \Delta_q h, \Delta_q h \rangle \right) \mathrm{vol}_q$$

- $e_{ij} = v_j - v_i$
- $dq_f = \begin{bmatrix} e_{01} \\ e_{02} \end{bmatrix}$
- $dh = \begin{bmatrix} h_1 - h_0 \\ h_2 - h_0 \end{bmatrix}$
- $g_f = \begin{bmatrix} |e_{01}|^2 & e_{01} \cdot e_{02} \\ e_{01} \cdot e_{02} & |e_{02}|^2 \end{bmatrix}$
- $\mathrm{vol}_f = \frac{1}{2} |e_{01} \times e_{02}|$
- $n_f = \frac{e_{01} \times e_{02}}{|e_{01} \times e_{02}|}$
- $(\Delta_q h)_v = \sum_{\substack{w|(v,w)\in E \\ \text{or } (w,v)\in E}} (\cot(\alpha_{vw}) + \cot(\beta_{vw}))(h_v - h_w)$
- $\mathrm{vol}_v = \frac{1}{3} \sum_{f|v\in f} \mathrm{vol}_f$



Discretization of (spatial) quantities (cont'd)

$$
\begin{aligned}
G_q^{a_0,a_1,b_1,c_1,d_1,a_2}(h,k) = \int_M & a_0 \langle h, k \rangle + a_1 g_q^{-1}(dh_m, dk_m) \\
& + b_1 g_q^{-1}(dh_+, dk_+) + c_1 g_q^{-1}(dh_\perp, dk_\perp) \\
& + d_1 g_q^{-1}(dh_0, dk_0) + a_2 \langle \Delta_q h, \Delta_q h \rangle \operatorname{vol}_q
\end{aligned}
$$

becomes

$$
\begin{aligned}
G_q^{a_0,a_1,b_1,c_1,d_1,a_2}(h,k) &= \sum_{v \in V} a_0 \langle h, k \rangle \operatorname{vol}_v \\
&+ \sum_{f \in F} \Big(a_1 g_f^{-1}(dh_m, dk_m) + b_1 g_f^{-1}(dh_+, dk_+) \\
&+ c_1 g_f^{-1}(dh_\perp, dk_\perp) + d_1 g_f^{-1}(dh_0, dk_0) \Big) \operatorname{vol}_f \\
&+ \sum_{v \in V} a_2 \left\langle \Delta_q h, \Delta_q k \right\rangle \operatorname{vol}_v
\end{aligned}
$$

Translating into code[1]

```
class DiscreteSurfaces(Manifold):

  def __init__(self, faces, equip=True):
    self.faces = faces
    # ...

  def belongs(self, point, atol=gs.atol):
    # check if point belongs to manifold

  def random_point(self, n_samples=1):
    # create random point
```

```
class ElasticMetric(RiemannianMetric):

  def __init__(
    self, space, a0, a1, b1, c1, d1, a2
  ):
    # ...

  def inner_product(self,
    tangent_vec_a, tangent_vec_b,
    base_point
  ):
    # compute inner product
```

```
space = DiscreteSurfaces(faces, equip=False)
space.equip_with_metric(ElasticMetric)

# point : array, shape=(..., n_vertices, 3)
space.metric.squared_dist(point_a, point_b)
```

[1] check out `geomstats.geometry.discrete_surfaces`

Riemannian geometry: inner product and distance

Discrete surfaces $(\mathfrak{M}, G)$:

$$\text{smooth manifold + inner product}$$

We have an **inner product** at each point q

$$G_q(\cdot,\cdot) \text{ on } T_q\mathfrak{M}$$

Distance as the norm of a tangent vector

$$d^2(q,p) = G_q(v,v), \text{where } v = \mathrm{Log}_q(p)$$

Back to code: Log[1]

```
space = DiscreteSurfaces(faces, equip=False)
space.equip_with_metric(ElasticMetric)

# point : array, shape=(..., n_vertices, 3)
space.metric.squared_dist(point_a, point_b)
```

is the same as:

```
tangent_vec = space.metric.log(point_b, point_a)
space.metric.inner_product(tangent_vec, tangent_vec, point_a)

# or
space.metric.squared_norm(tangent_vec, point_a)
```

- code is slow? log solver is the culprit!

[1] check out `geomstats.geometry.riemannian_metric`

Back to code: Log[1]

```python
space = DiscreteSurfaces(faces, equip=False)
space.equip_with_metric(ElasticMetric)

# point : array, shape=(..., n_vertices, 3)
space.metric.squared_dist(point_a, point_b)
```

is the same as:

```python
tangent_vec = space.metric.log(point_b, point_a)
space.metric.inner_product(tangent_vec, tangent_vec, point_a)

# or
space.metric.squared_norm(tangent_vec, point_a)
```

- code is slow? log solver is the culprit!

[1] check out `geomstats.geometry.riemannian_metric`

Geodesic boundary value problem

Find **paths** of shortest length connecting p_0 and p_1.

Induced geodesic distance:

$$d_G(p_0, p_1) = \inf_{p \in \mathcal{P}_{p_0}^{p_1}} L_G(p)$$
$$= \inf_{p \in \mathcal{P}_{p_0}^{p_1}} \int_0^1 \sqrt{G_{p(t)}\left(\partial_t p(t), \partial_t p(t)\right)} dt$$

with:

$$\mathcal{P}_{p_0}^{p_1} := \{p \in C^\infty([0,1], \mathfrak{M}) : p(0) = p_0, p(1) = p_1\}$$

Geodesic boundary value problem

Find **paths** of shortest length connecting p_0 and p_1.

Induced geodesic distance:

$$d_G\left(p_0, p_1\right) = \inf_{p \in \mathcal{P}_{p_0}^{p_1}} L_G(p)$$
$$= \inf_{p \in \mathcal{P}_{p_0}^{p_1}} \int_0^1 \sqrt{G_{p(t)}\left(\partial_t p(t), \partial_t p(t)\right)} dt$$

with:

$$\mathcal{P}_{p_0}^{p_1} := \{p \in C^\infty([0,1], \mathfrak{M}) : p(0) = p_0, p(1) = p_1\}$$

Path

Let $q : [0,1] \to \mathfrak{M}$ denote a path of triangular meshes.

In practice, we discretize:

$$V_i = q(t_i), \quad t_i = \frac{i}{N}, i = 0, \cdots, N$$

Using finite differences,

$$\dot{V}_i = N\left(V_{i+1} - V_i\right)$$

Back to code: geodesic[1]

```
# q : callable
# float -> array, shape=(..., n_vertices, 3)
q = space.metric.geodesic(point_a, point_b)

geod_point = q(0.5)

# behind the scenes
# q: UniformlySampledDiscretePath
# V: array, shape=(..., n_grid, n_vertices, 3)
V = q.interpolator.data
```

```
class UniformlySampledDiscretePath:
  def __init__(self, path, interpolator):
    # ...

  def __call__(self, t):
    # ...


class UniformUnitIntervalLinearInterpolator:
  def __init__(self, data, point_ndim):
    # ...

  def interpolate(self, t):
    # ...
```

[1] check out `geomstats.numerics.geodesic/path/interpolation`

Riemannian energy: time discretization

Let $q : [0,1] \to \mathfrak{M}$ denote a path of triangular meshes.

$$E(q) := \frac{1}{2}\int_0^1 G_{q(t)}(\dot{q}(t), \dot{q}(t))dt$$

Discrete counterpart:

$$E(V) = \frac{1}{2N}\sum_{i=0}^{N-1} G_{V_i}\left(\dot{V}_i, \dot{V}_i\right)$$

Back to code: Riemannian energy[1]

```
# q : callable
# float -> array, shape=(..., n_vertices, 3)
q = space.metric.geodesic(point_a, point_b)

# array, shape=(..., n_grid, n_vertices, 3)
V = q.interpolator.data

path_energy = UniformlySampledPathEnergy(space)

path_energy(V)
```

```
class UniformlySampledPathEnergy:
  def __init__(self, space):
    # ...

  def __call__(self, path):
    # returns path energy
    # ...

  def energy_per_time(self, path):
    # handles finite differences
    # ...
```

[1] check out `geomstats.numerics.path`

Path straightening

Find discrete path V that minimizes Riemannian energy $E(V)$:

$$V^* = \arg\min_V E(V)$$

Can be solved with **gradient**-based algorithms.

Algorithm 1 Geodesic BVP for Parameterized Surfaces

Require:

V0, V1 : source and target surfaces

V : initial guess for discrete path

cost(V) $\leftarrow$ E([V0, V, V1])

V $\leftarrow$ L-BFGS(V, cost)

Back to code: path straightening[1]

```
# q : callable
# float -> array, shape=(..., n_vertices, 3)
q = space.metric.geodesic(point_a, point_b)
```

is the same as:

```
log_solver = PathStraightening(
  space, path_energy, n_nodes, optimizer
)

log_solver.geodesic_bvp(point_b, point_a)
```

```
class PathStraightening(LogSolver):
  def discrete_geodesic_bvp(
    self, point, base_point,
  ):
    # returns energy-minimizing discrete path

  def geodesic_bvp(self, point, base_point):
    # calls discrete_geodesic_bvp
    # returns UniformlySampledDiscretePath

  def log(self, point, base_point):
    # calls discrete_geodesic_bvp
    # returns tangent vector
```

[1] check out `geomstats.numerics.geodesic`

More code: optimizer[1]

```
class ScipyMinimize(Minimizer):
  def __init__(
    self,
    method,
    autodiff_jac,  # bool
    # ...
  ):
    # ...

  def minimize(self, fun, x0):
    # wraps scipy.optimize.minimize
```

Alternatives: `TorchLBFGS`, `TorchminMinimize`

[1] check out `geomstats.numerics.optimization`

Zooming out

- from an **inner product**, we get **distances** and **geodesics** by solving an **optimization problem**
- optimization problem finds **position of vertices** of meshes in a path

NB: most of the introduced objects are **generic**
(e.g. `PathStraightening`)

Zooming out

- from an **inner product**, we get **distances** and **geodesics** by solving an **optimization problem**
- optimization problem finds **position of vertices** of meshes in a path

NB: most of the introduced objects are **generic**
(e.g. `PathStraightening`)

Zooming out

- from an **inner product**, we get **distances** and **geodesics** by solving an **optimization problem**
- optimization problem finds **position of vertices** of meshes in a path

NB: most of the introduced objects are **generic**
(e.g. `PathStraightening`)

Dive in warning

Main goal:

introduce **unparameterized surfaces**

- introduce **varifold** distance and **kernels**
- present strategy to solve **relaxed alignment problem**

Unparameterized surfaces

Set of triangle meshes ~~with~~ ~~**fixed combinatorial structure**~~ $\cancel{\mathfrak{M}}$.

- ~~fixed number of vertices~~ $\cancel{V}$
- ~~fixed connectivities: fixed set of edges~~ $\cancel{E}$ ~~and faces~~ $\cancel{F}$

Each point is an **equivalence class**:

$$[q] = \{q \circ \varphi : \varphi \in \mathcal{D}\}$$

Implementation heavily relies on **parameterized surfaces.**

Relaxed alignment problem

$$d_G([p_0], [p_1]) = \inf_{\varphi \in \mathcal{D}} \inf_{p \in \mathcal{P}_{p_0}^{\varphi \cdot p_1}} L_G(p)$$

Can be written as:

$$\min_{\tilde{p}_1 \in \mathfrak{M}_0} d_G(p_0, \tilde{p}_1) \quad \text{subject to} \quad d_G(\tilde{p}_1, [p_1]) = 0$$

Or, more interestingly:

$$\arg\min_{\tilde{p}_1 \in \mathfrak{M}_0} E_G(p) + \lambda \Gamma(p(1), p_1), \text{where } p \in \mathcal{P}_{p_0}^{\tilde{p}_1}$$

i.e. $\Gamma(p(1), p_1)$ enforces orbit membership.

Relaxed alignment problem

$$d_G\left([p_0], [p_1]\right) = \inf_{\varphi \in \mathcal{D}} \inf_{p \in \mathcal{P}_{p_0}^{\varphi \cdot p_1}} L_G(p)$$

Can be written as:

$$\min_{\tilde{p}_1 \in \mathfrak{M}_0} d_G\left(p_0, \tilde{p}_1\right) \quad \text{subject to} \quad d_G\left(\tilde{p}_1, [p_1]\right) = 0$$

Or, more interestingly:

$$\arg\min_{\tilde{p}_1 \in \mathfrak{M}_0} E_G(p) + \lambda \Gamma\left(p(1), p_1\right), \text{where } p \in \mathcal{P}_{p_0}^{\tilde{p}_1}$$

i.e. $\Gamma\left(p(1), p_1\right)$ enforces orbit membership.

Relaxed alignment problem

$$d_G\left([p_0], [p_1]\right) = \inf_{\varphi \in \mathcal{D}} \inf_{p \in \mathcal{P}_{p_0}^{\varphi \cdot p_1}} L_G(p)$$

Can be written as:

$$\min_{\tilde{p}_1 \in \mathfrak{M}_0} d_G\left(p_0, \tilde{p}_1\right) \quad \text{subject to} \quad d_G\left(\tilde{p}_1, [p_1]\right) = 0$$

Or, more interestingly:

$$\arg\min_{\tilde{p}_1 \in \mathfrak{M}_0} E_G(p) + \lambda \Gamma\left(p(1), p_1\right), \text{where } p \in \mathcal{P}_{p_0}^{\tilde{p}_1}$$

i.e. $\Gamma\left(p(1), p_1\right)$ enforces orbit membership.

Varifold distance

Where can we find such Γ? **Varifold theory** comes to rescue!

A **varifold distance** (between two surfaces) is blind to reparameterizations.

In other words, it is $\mathcal{D} \times \mathcal{D}$-invariant:

$$d_V\left(p_0, p_1\right) = d_V\left(p_0 \circ \varphi_0, p_1 \circ \varphi_1\right)$$

Therefore, we can use

$$\Gamma\left(p(1), p_1\right) = d_V^2\left(p(1), p_1\right)$$

Varifold distance

Where can we find such Γ? **Varifold theory** comes to rescue!

A **varifold distance** (between two surfaces) is blind to reparameterizations.

In other words, it is $\mathcal{D} \times \mathcal{D}$-invariant:

$$d_V\left(p_0, p_1\right) = d_V\left(p_0 \circ \varphi_0, p_1 \circ \varphi_1\right)$$

Therefore, we can use

$$\Gamma\left(p(1), p_1\right) = d_V^2\left(p(1), p_1\right)$$

Varifold distance

Where can we find such Γ? **Varifold theory** comes to rescue!

A **varifold distance** (between two surfaces) is blind to reparameterizations.

In other words, it is $\mathcal{D} \times \mathcal{D}$-invariant:

$$d_V\left(p_0, p_1\right) = d_V\left(p_0 \circ \varphi_0, p_1 \circ \varphi_1\right)$$

Therefore, we can use

$$\Gamma\left(p(1), p_1\right) = d_V^2\left(p(1), p_1\right)$$

Kernels

$$\langle q_0, q_1 \rangle_V = \sum_{f_0 \in F_0} \sum_{f_1 \in F_1} K\left(c_{f_0}, c_{f_1}, n_{f_0}, n_{f_1}\right) \mathrm{vol}_{f_0} \, \mathrm{vol}_{f_1}$$

Kernel factorization (positional $\times$ spherical):

$$K\left(c_{f_0}, c_{f_1}, n_{f_0}, n_{f_1}\right) = K_p(c_{f_0}, c_{f_1}) K_s(n_{f_0}, n_{f_1})$$

Kernels

$$\langle q_0, q_1 \rangle_V = \sum_{f_0 \in F_0} \sum_{f_1 \in F_1} K\left(c_{f_0}, c_{f_1}, n_{f_0}, n_{f_1}\right) \mathrm{vol}_{f_0} \mathrm{vol}_{f_1}$$

Kernel factorization (positional $\times$ spherical):

$$K\left(c_{f_0}, c_{f_1}, n_{f_0}, n_{f_1}\right) = K_p(c_{f_0}, c_{f_1}) K_s(n_{f_0}, n_{f_1})$$

(Some) kernels[1]

Positional:

- Gaussian: $K_p(x,y) = e^{-\|x-y\|^2/\sigma^2}$
- Cauchy: $K_p(x,y) = \frac{1}{1+\|x-y\|^2/\sigma^2}$

Spherical:

- Linear: $K_s(u,v) = \langle u,v \rangle$
- Binet: $K_s(u,v) = \langle u,v \rangle^2$
- "Squared restricted Gaussian": $K_s(u,v) = e^{-2\langle x,y \rangle^2/\sigma^2}$
- Restricted Gaussian: $K_s(u,v) = e^{-2\langle x,y \rangle/\sigma^2}$

Kernel addition results in a valid kernel.

[1] Charon et al., “12 - Fidelity metrics between curves and surfaces”, 2020.

Back to code: kernels and varifold metric[1,2]

```
position_kernel = GaussianKernel(sigma=1.0, init_index=0)
tangent_kernel = BinetKernel(
  init_index=position_kernel.new_variable_index()
)
kernel = SurfacesKernel(
  position_kernel,
  tangent_kernel,
  signal_kernel=None,
)

varifold_metric = VarifoldMetric(kernel)
```

```
class SurfacesKernel:
  def __call__(self, point_a, point_b):
    # evaluates kernel
    # ...


class VarifoldMetric:
  def scalar_product(self, point_a, point_b):
    # simply calls kernel
    # ...

   def squared_distance(self, point_a, point_b):
    # ...

   def loss(self, target_point):
    # outputs a callable
    # ...
```

[1] check out `geomstats.varifold`
[2] thank you `pykeops` team!

Oriented varifolds

Oriented varifolds[1] generalize:

- a model of measures for point clouds[2]: $K_s(u, v) = 1$
- currents[3]: $K_s(u, v) = \langle u, v \rangle$
- varifolds[4]: orientation-invariant kernel
  - $K(u, v) = K(u, -v) = K(-u, v)$
  - e.g. $K_s(u, v) = \langle u, v \rangle^2$

---

[1] Kaltenmark, Charlier, and Charon, "A General Framework for Curve and Surface Comparison and Registration With Oriented Varifolds", 2017.

[2] Glaunes, Trouve, and Younes, "Diffeomorphic matching of distributions", 2004.

[3] Vaillant and Glaunès, "Surface Matching via Currents", 2005.

[4] Charon and Trouvé, "The Varifold Representation of Nonoriented Shapes for Diffeomorphic Registration", 2013.

Back to code: surface[1]

```
varifold_metric = VarifoldMetric(kernel)

point_a = Surface(vertices_a, faces_a)
point_b = Surface(vertices_b, faces_b)

varifold_metric.squared_dist(point_a, point_b)
```

```
class Surface:
  def __init__(
    self,
    vertices,
    faces,
    signal=None
  ):
    # ...
    self.face_centroids = # ...
    self.face_normals = # ...
    self.face_areas = # ...
```

[1] check out `geomstats._mesh`

More code: relaxed path straightening[1]

```
log_solver = RelaxedPathStraightening(
  space,
  path_energy,
  n_nodes,
  optimizer,
  discrepancy_loss,
)

log_solver.geodesic_bvp(point_b, point_a)
```

```
class RelaxedPathStraightening(LogSolver, AlignerAlgorithm):
  def discrete_geodesic_bvp(
    self, point, base_point,
  ):
    # returns energy-minimizing discrete path

  def geodesic_bvp(self, point, base_point):
    # returns UniformlySampledDiscretePath

  def log(self, point, base_point):
    # returns tangent vector

  def align(self, point, base_point):
    # aligns point to base point
```

[1] check out `geomstats.geometry.discrete_surfaces`

More code: relaxed path straightening (cont'd)[1]

```
class RelaxedPathStraightening(LogSolver, AlignerAlgorithm):
  def __init__(
    total_space,
    n_nodes=3,
    lambda_=1.0,
    discrepancy_loss=None,
    path_energy=None,
    optimizer=None,
    initialization=None,
  ):
    # ...
```

[1] check out `geomstats.geometry.discrete_surfaces`

More code: quotient structure[1,2]

```
log_solver = RelaxedPathStraightening(
  space, path_energy, n_nodes, optimizer
)

log_solver.geodesic_bvp(point_b, point_a)
```

is the same as:

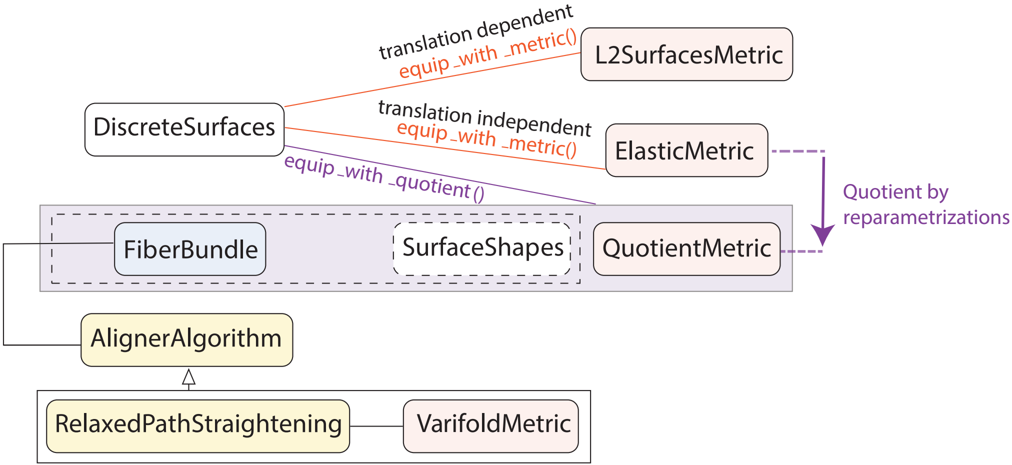
```
space = DiscreteSurfaces(faces)
space.equip_with_metric(ElasticMetric)
space.equip_with_group_action("reparametrizations")
space.equip_with_quotient_structure()

space.quotient.metric.geodesic(base_point, point)
```

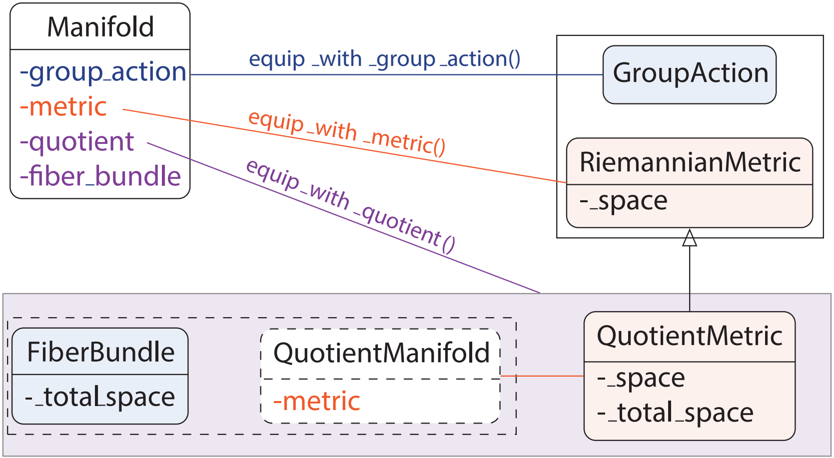
[1] Pereira et al., Learning from landmarks, curves, surfaces, and shapes in Geomstats, 2024.

[2] check out `geomstats.geometry.discrete_surfaces`

Quotient structure: surfaces



(Abstract) quotient structure



Applies to **landmarks**, **curves**, **surfaces**, **full-rank correlation matrices**.

Zooming out

- **parameterized surfaces** and **varifold distance** can be leveraged for computations with **unparameterized surfaces**

Composing objects

- objects presented can be seen as **building blocks** to more complex algorithms
- e.g. `MultiresPathStraightening` may use `PathStraightening` at each iteration
  - not done yet
  - good first contribution
- `RelaxedPathStraightening` can be used to implement a `LambdaAdaptiveRelaxedPathStraightening`
  - see unofficial implementation in `polpo.registration.surface`

Composing objects

- objects presented can be seen as **building blocks** to more complex algorithms
- e.g. `MultiresPathStraightening` may use `PathStraightening` at each iteration
  - not done yet
  - good first contribution
- `RelaxedPathStraightening` can be used to implement a `LambdaAdaptiveRelaxedPathStraightening`
  - see unofficial implementation in `polpo.registration.surface`

Composing objects

- objects presented can be seen as **building blocks** to more complex algorithms
- e.g. `MultiresPathStraightening` may use `PathStraightening` at each iteration
  - not done yet
  - good first contribution
- `RelaxedPathStraightening` can be used to implement a `LambdaAdaptiveRelaxedPathStraightening`
  - see unofficial implementation in `polpo.registration.surface`

Back to code: multiresolution path straightening[1]

```
log_solver = PathStraightening(
  space, path_energy, n_nodes, optimizer
)

geod = None
for n_nodes in n_nodes_ls:
  log_solver.n_nodes = n_nodes
  if geod is not None:
    times = gs.linspace(0, 1, n_nodes)
    log_solver.initialization = geod(times)[1:-1]

  geod = log_solver.geodesic_bvp(point, base_point)
```

*needs slight adaptations to actually work

[1] check out `geomstats.numerics.geodesic`

Wrapping up

- a lot of objects
- users do not necessarily need to be aware of them
- performance is tied to choice of numerical parameters
- reusability of objects makes long-term development easier and more reliable

Thank you for your attention!

https://geomstats.ai/
https://github.com/geomstats/geomstats

Nicolas Guigui, Nina Miolane, Xavier Pennec. 2022. **Introduction to Riemannian Geometry and Geometric Statistics: from basic theory to implementation with Geomstats**. Foundations and Trends in Machine Learning.

Charon, Nicolas and Alain Trouvé. "The Varifold Representation of Nonoriented Shapes for Diffeomorphic Registration". In: SIAM Journal on Imaging Sciences 6.4 (Jan. 2013). Publisher: Society for Industrial and Applied Mathematics, pp. 2547–2580. DOI: 10.1137/130918885. URL: https://epubs.siam.org/doi/abs/10.1137/130918885.

Charon, Nicolas et al. "Fidelity metrics between curves and surfaces: currents, varifolds, and normal cycles". en. In: Riemannian Geometric Statistics in Medical Image Analysis. Ed. by Xavier Pennec, Stefan Sommer, and Tom Fletcher. Academic Press, Jan. 2020, pp. 441–477. ISBN: 978-0-12-814725-2. DOI: 10.1016/B978-0-12-814725-2.00021-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128147252000212.

Crane, Keenan. "Discrete differential geometry: An applied introduction". 2018. URL: https://www.cs.cmu.edu/~kmcrane/Projects/DDG/paper.pdf.

Glaunes, J., A. Trouve, and L. Younes. "Diffeomorphic matching of distributions: a new approach for unlabelled point-sets and sub-manifolds matching". In: Proceedings of the 2004 IEEE Computer Society Conference on C Vol. 2. ISSN: 1063-6919. June 2004, pp. II–II. DOI: 10.1109/CVPR.2004.1315234. URL: https://ieeexplore.ieee.org/document/1315234.

Guigui, Nicolas, Nina Miolane, and Xavier Pennec. "Introduction to Riemannian Geometry and Geometric Statistics: from basic theory to implementation with Geomstats". en. In: Foundations and Trends in Machine Learning (2022). URL: https://hal.inria.fr/hal-03766900.

Hartman, Emmanuel et al. "Elastic Shape Analysis of Surfaces with Second-Order Sobolev Metrics: A Comprehensive Numerical Framework". en. In: International Journal of Computer Vision 131.5 (May 2023), pp. 1183–1209. ISSN: 1573-1405. DOI: 10.1007/s11263-022-01743-0. URL: https://doi.org/10.1007/s11263-022-01743-0.

Kaltenmark, Irene, Benjamin Charlier, and Nicolas Charon. "A General Framework for Curve and Surface Comparison and Registration With Oriented Varifolds". In: 2017, pp. 3346–3355. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Kaltenmark_A_General_Framework_CVPR_2017_paper.html.

Pereira, Luís F. et al. Learning from landmarks, curves, surfaces, and shapes in Geomstats. arXiv:2406.10437 [cs]. June 2024. DOI: 10.48550/arXiv.2406.10437. URL: http://arxiv.org/abs/2406.10437 (visited on 02/10/2025).

Vaillant, Marc and Joan Glaunès. "Surface Matching via Currents". en. In: Information Processing in Medical Imaging. Ed. by Gary E. Christensen and Milan Sonka. Berlin, Heidelberg: Springer, 2005, pp. 381–392. ISBN: 978-3-540-31676-3. DOI: 10.1007/11505730_32.