SHORT USER GUIDE: PY-LDDMM

LAURENT YOUNES

This library implements a series of functions in python around the LDDMM algorithm for curve, surface and image matching. This user guide (like the library) is work in progress and will be progressively updated.

1. Requirements

The following python packages are used in the current version: numpy, scipy, vtk, numba, pyfftw, matplotlib, pillow, scikit-image, nibabel, imageio, h5py, tqdm, pandas, setuptools, pykeops

2. Basic structures

This is a quick description of various structures for which registration routines are provided.

2.1. **Point sets.** Point sets are simply N by d vectors of points, where d is the dimension and N the number of points. A few I/O functions are provided in pointSets.py.

2.2. **Curves.** Curves are represented as as a set of vertices (an N by d array, where d is the dimension), and a set of edges, represented as an M by 2 integer array, where each row defined an oriented edge between two vertices. Various constructor functions and I/O methods are provided in curves.py.

2.3. **Surfaces.** Surfaces are represented as as a set of vertices (an N by d array, where d is the dimension), and a set of faces, represented as an M by 3 integer array, where each row defined an oriented triangle. Various constructor functions and I/O methods are provided in surfaces.py.

2.4. **Surface sections.** A surface section is a 3D curve supported by a plane. The file surface_sections.py defines a class containing a family of such sections.

2.5. **Meshes.** Meshes are represented as as a set of vertices (an N by d array, where d is the dimension), and a set of faces, represented as an M by $d+1$ integer array, where each row defined an oriented simplex. They also contain an "image" field that enable varifold image comparison. Various constructor functions and I/O methods are provided in surfaces.py.

2.6. **Diffeomorphisms.** The file diffeo.py provides several tools for dealing with functions discretized on grids, including interpolation, smoothing and composition.

3. Distances

Distances serving as data attachment terms are defined between the previous structures. All distances are implemented as:

$$\text{distance}(\text{shape}, \text{target}, \text{args}),$$

which returns a scalar. The first variable ("shape") usually changes from call to call while the other are fixed within a given run of LDDMM. To avoid unnecessary computation, this function is decomposed in the form

$$\text{distance\_def}(\text{shape}, \text{target}, \text{args}) + \text{distance0}(\text{target}, \text{args})$$

where distance0 computes the part of the distance that does not depend on "shape".

Finally, for each type of distance, a function

$$\text{distance\_grad}(\text{shape}, \text{target}, \text{args})$$

that returns the gradient of the distance function with respect to the first variable, is also implemented. In most cases, the implementation of these functions is all that needs to be done to develop "new versions" of LDDMM.

4. Kernel

The Kernel class is defined in "kernelFunctions.py". The constructor is (listing only the main parameters):

```
def __init__(self,
                        name='gauss',
                        sigma = (1.,),
                        order = 3, [...]):
```

"Name" is the kernel's name (currently, either "gauss" or "laplacian"). "sigma" provides the kernel width(s), either a scalar or a list (for multiple kernels). "order" is the order for Laplacian kernels (must be less than or equal to 4).

5. Evolution equations

Several types of evolution equations for point sets and other structures are implemented in files such as pointEvolution.py. We skip the details, since these files should only be modified by domain experts.

6. Matching

Multiple classes of matching functions are included in the package, depending on the type of objects that are compared. Recall that LDDMM minimizes

$$\int_0^1 \|v(t)\|^2 dt + \frac{1}{\sigma^2}\text{dist}(\text{deformed template}, \text{target})$$

where the deformed template is the result of applying the value at time 1 of the solution of $\partial_t \varphi = v(t, \varphi)$ to the template. The second term is referred below as the "error term".

6.1. **Generic call.** Matching objects are created with the call

```
matchingObject(Template, Target, options)
```

where Template and Target are the two compared objects. Several types of maching objects are defined in the package, including poinsetMatching, curveMatching, surfaceMatching, meshMatching, imageMatching and variations.

6.2. **options.** The field options is a dictionary including the many parameters used by the code. The following lists the most important among them

- *timeStep* is the geodesic "time" discretization length. The geodesic is estimated over the unit interval and the number of time steps is ceil($1/$*timeStep*).
- *KparDiff* is the kernel applied to vector fields in LDDMM. It is specified either as an instance of *class Kernel* or as a list in the form (type, width, order).
- *KparDist* is the kernel for distances based on currents, varifolds or measures. It is specified in the same way as *KparDiff*.
- *sigmaError* is the normalization for error term (the parameter σ above).
- *errorType* defines the type of distance that is used in the error term, such as 'measure', 'varifold', 'current'...
- *algorithm* defines the first-order optimization scheme, and can be either 'bfgs' or 'cg' (for conjugate gradient).
- *mode* 'quiet', 'normal' or 'debug', determines the amount of output. Debug makes additional tests, including on the correctness of gradients and forces GPU computation to work at 64 bits (resulting in lower execution).
- *maxIter*: maximum number of iterations.
- *internalWeight*: Weight in front of the internal cost.
- *internalCost*: object dependent keyword specifying additional terms (e.g., elastic) in the energy function.