IMODAL: creating learnable user-defined deformation models Appendix

Leander Lacroix
INSERM U1299, Université Paris-Saclay
leander.lacroix@protonmail.com

Benjamin Charlier
IMAG
Univ. Montpellier, France
benjamin.charlier@umontpellier.fr

Alain Trouvé
Centre Borelli, ENS Paris-Saclay
alain.trouve@ens-paris-saclay.fr

Barbara Gris
LJLL, Sorbonne Université, CNRS
gris@ljll.math.upmc.fr

A. Expression of the field generator for implicit deformation modules

A.1. Generic solution

Let $Y = \mathbb{R}^n$ and $S_q \in L(V, \mathbb{R}^n)$ and $A_q \in L(H, \mathbb{R}^n)$ continuous linear operators. For any h, since $J(v) = |v|_V^2 + \frac{1}{\nu}|S_q(v) - A_q(h)|^2$ is convex and coercive, we have a unique solution $\zeta_q(h)$ that can be calculated using the lagrangian $\mathcal{L} : V \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

$$\mathcal{L}(v, w, \lambda) = |v|_V^2 + \frac{|w|^2}{\nu} + 2\lambda^T(A_q(h) - (S_q(v) + w))$$

giving from the Euler-Lagrange equations $\partial_v \mathcal{L} = \partial_w \mathcal{L} = \partial_\lambda \mathcal{L} = 0$ the equations $\zeta_q(h) = KS_q^*\lambda$, $w = \nu\lambda$ and $(S_qKS_q^* + \nu I)\lambda = A_q(h)$. We deduce $\lambda = (S_qKS_q^* + \nu I)^{-1}A_q(h)$, $\zeta_q(h) = KS_q^*\lambda$ and $c_q(h) = \lambda^T S_qKS_q^*\lambda + \nu|\lambda|^2 = \lambda^T(S_qKS_q^* + \nu I)\lambda = A_q(h)^T(S_qKS_q^* + \nu I)^{-1}A_q(h)$. The invertibility of the operator $S_qKS_q^* + \nu I$ is deduced from the surjectivity of S_q and the positivity of $S_qKS_q^*$.

A.2. Implicit module of order 0

Let us detail this computation for an implicit deformation module of order 0. In this case, the space of geometrical descriptors is $\mathcal{O} = (\mathbb{R}^d)^N$ and the space of controls is $H = (\mathbb{R}^d)^N$. The constraint operator is $S_q : v \in V \mapsto (v(q_1), \dots, v(q_N)) \in (\mathbb{R}^d)^N$ and the linear operator A_q is defined by $A_q(h) = (K_q + \nu Id)h$. Then $KS_q^* = \sum_i K(x_i, \cdot)h_i$ so $S_qKS_q^* = K_{q,q}h$, and $\lambda = (S_qKS_q^* + \nu Id)^{-1}A_q(h) = h$. In addition, the cost is then $c_q(h) = h^T(K_{q,q} + \nu Id)h$. Note that for the operator S_q to be surjective, we need to consider geometrical descriptors $= (x_1, \dots, x_N)$ such that $x_i \neq x_j$ for $j \neq i$ (which will be the case in practice).

B. IMODAL

This framework is implemented in IMODAL (Implicit Modular Object Deformation Analysis Library), in Python, using the librairies PyTorch[16] for automatic differentiation and KeOps [7] to avoid memory overflow in GPU computations. It is a *modular* implementation in the sense that it relies on several abstract classes that one can combine in order to design a deformation model adapted to the observed data.

B.1. Deformation Modules

This library relies on the abstract class `DeformationModules` which implements the notion of deformation module as defined in Definition 1. A large variety of deformation modules (in particular the ones presented in Section 2.3) are implemented as subclasses of the abstract one `DeformationModules`. In order to use them, one only needs to specify their specific parameters such as the scale σ of the scalar Gaussian kernel K_σ (examples of Section 2.3.1, 2.3.2, 2.3.3). In the specific case of implicit modules of order 1 (Section 2.3.3), it is necessary to specify the growth model operator, $h \mapsto (D_i(h))_{i \in I}$, namely a (growth model) tensor C of size $N \times d \times p$ with N the number of points on which the field has to be constrained, d the dimension of the ambient space and p the dimension of the control space so that $D_i(h) = \mathrm{diag}(C[i]h)$ for $1 \leq i \leq N$.

The main attributes of the `DeformationModules` class are the control h (implemented as a tensor) and the geometrical descriptor q. This latter is implemented via a class called `Manifold` which is in particular defined via the way vector fields can act on it (defining the infinitesimal action $v \cdot q$). Unlike the parameters defining the deformation modules, these attributes evolve during the integration

of the shooting equation 4.

An important function of the module class is `field_generator` which generates, given the current values of geometrical descriptor q and control h, the vector field $\zeta_q(h)$.

The combination of deformation modules (Definition 2) is implemented via a particular subclass of `DeformationModules`, named `CompoundModule`, which is defined from a list of `DeformationModules`. This `CompoundModule` subclass allows to build easily complex deformation models as the superposition of several simple ones.

B.2. Data

Several types of data can be used such as point clouds, unparametrized curves, unparametrized meshes and images (the interpolation of images is based on the code developed in Deformetrica [6]), they are implemented as subclasses of the abstract `Deformable` class. Each one has a function which generates the corresponding silent deformation module (see Section 2.3.4 and Remark 5) so it does not need to be defined manually.

Several attachment terms are defined as subclasses of the abstract `Attachment` such as Euclidean distances for point clouds, varifold attachment [8] (based on the code developed in KeOps [7] and Deformetrica [6]) for unparametrized curves and meshes, and L^2 distance for images. In the case where the observed data is formed of several parts (for example a curve and a point cloud such as in the Example 3.1), it is possible to consider them both simultaneously, the total attachment term being then a weighted sum of several ones (each corresponding to one part).

B.3. Models

The class `RegistrationModel` is the keystone allowing to perform a registration using a chosen deformation model. It is initialized with lists of `Deformables` (coding for the observed data), `Attachment` (defining the attachment functions for these data) and `DeformationModules` (defining the deformation model). This class enables to compute, given target `Deformables` and initial values for the geometrical descriptors and momenta of the deformation modules, the functional 5. In order to minimize this functional, a `Fitter` class is created from `RegistrationModel` and runs the optimisation. It interfaces most PyTorch and Scipy optimizers along with a gradient descent with linear search optimization algorithm.

It is possible to estimate the parameters of the deformation modules with a key-word `other_parameters` in `RegistrationModel`. It is also possible to add a callback function that is evaluated before model evaluation. This allows us to estimate some parameters of the model as functions of meta-parameters which can be estimated. This is for instance used in Section 3.1 in order to estimate the growth factor as a polynomial of the point coordinates (the parameters of the polynomial are estimated, not directly the growth factor).

B.4. Utilities

In order to facilitate the definition of the deformation model, several functions from the `Utility` toolbox can be used. In particular, it is possible, from a shape (curve in 2D or mesh in 3D for instance) to define *automatically* a regular grid of points around the shape. This is particularly useful to define the geometrical descriptors of implicit deformation modules.

B.5. GPU

It is possible to choose to run the code on CPU or on GPU (using KeOps [7] to avoid memory overflow if necessary and accelerate computations).