Conversion to Python by Flatiron Institute

August 2024

1 From Bob Carpenter email

To summarize, the plan is to create a simple functional design document for the HDM package (in a new repo in the tooth-and-claw organization) that specifies the inputs and outputs. I say "functional" here because I don't want to get hung up in writing a technical spec yet. A functional spec explains what is getting computed whereas a technical spec will explain how that's done. Of course, a functional spec needs to specify something that's technically feasible.

The functional spec should list all the classes and functions that will be used by the package, and for each one, provide a detailed signature with the inputs and outputs as well as a precise description of what is computed (but not the algorithm or code to do it). Any constraints on the inputs, either individually or collectively, should be part of the spec. Some idea of sizes that people will use will really help here and also any performance expectations in terms of speed.

Ideally, the spec will be detailed enough to independently write the code and the top-level unit tests.

We can start with just the list of inputs and outputs, then translate it into Python. I like to keep everything nicely typed and we use Mypy for our projects to document types. And black to auto-format.

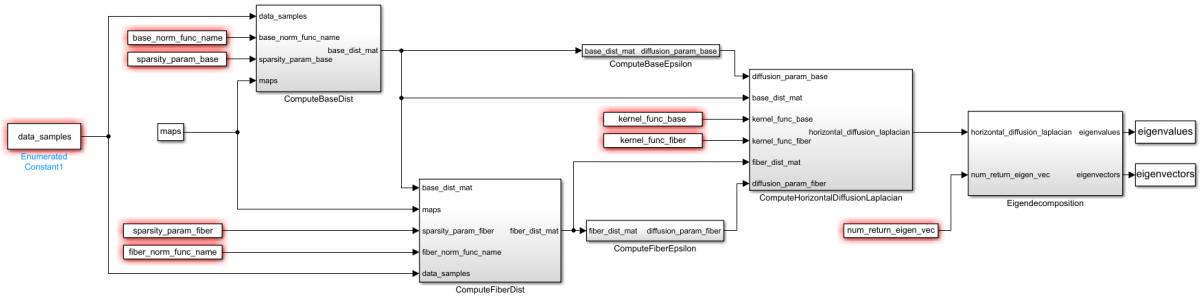
The spec should also list the intended clients. Will other pieces of software link to this? Will people just run it standalone? Will it get run on a cluster? Will the clients have access to a GPU and if so, how much memory will it have (the $50KGPUshave80GB, the$2K ones have on the order of 10GB)? Rob had mentioned he'd been targeting low-level compute that someone might have on their desk.

The tentative plan is to

1. use in-memory data structures rather than file formats, and
2. target users with decent amounts of memory and a GPU (e.g., on a cluster)
3. not introduce dependencies on the big ML packages like PyTorch or JAX
4. allow call outs to C++ on the back end
   Thought we didn't talk about it, presumably the plan also includes
5. releasing the package to the public on pypi and the docs on readthedocs
   I also believe the plan was to:
6. release under the BSD-3 license (MIT would be similar)

You'll want to create the repo with the license. Then everyone who submits anything agrees to release what they submit under the license as part of the GitHub End User License Agreement (EULA).

- Bob



P.S. Here's the repo where we keep the functional design docs for Stan. The top-level README goes over what there is of the process:

https://github.com/stan-dev/design-docs

The ones that are merged are the ones that have been accepted and you can see those here:

https://github.com/stan-dev/design-docs/tree/master/designs

Keep in mind that Stan's a much bigger project with many more devs, so we have to take things to this level. We don't need something quite so formal for HDM working with a small group on a new project.

1.1 Meeting Sep 3

Bob would September 19th at 10am work for you for the next meeting?

For action items before our next meeting I think it came down to

1. Figure out the sub functions the larger function can be broken down into. From the functional spec it seems like steps 1, 2, and 3 are good place to start thinking about that. 2. Think about how to represent the sparse block structure of the correspondence matrix. Ingrid I got a little confused about when you mentioned slicing on one index will change how you access the other matrix.

Bob: Sounds good. We also want to start breaking this down into Python data structures. For example, the callback functions are presumably Python functions with a signature that could be expressed in the spec. This will require deciding how general to make the API (e.g., np.array vs. ArrayLike in mypy typing). The current write-up's a great start for this.

2 Functional design spec

— output: list of eigenvalues list of eigenvalues —

2.1 HDM Functional Specification

2.2 Goals

- Clean interface and implementation of the Horizontal Diffusion Maps (HDM).

2.3 Design considerations

2.4 Assumptions

2.5 Workflow

2.6 HDM API

The HDM interface will be implemented as a Python package with the following functions and classes.

3 Functions

3.1 ComputeBaseDist

ComputeBaseDist(data_samples, base_norm_function_name = 'EuclideanNorm', sparsity_param_base = 10, data_object_maps, subsample_mapping = 1)

Construct diffusion matrix B ($K \times K$ matrix) on the base manifold.

1. If 'subsample_mapping' is not 1, sample each 'data_samples' object with respect to the given 'subsample_mapping' percentage, and create a subsampled version of the 'data_samples'.
2. For the (i, j)th entry in 'base_dist_mat'
   - Set B_{ij} by applying the 'base_norm_function_name' function to i, and j objects from the 'data_samples', while taking into account the 'data_object_maps'
   - if B_{ij} is larger than the base sparsity radius ('base_dist_mat'), then set $B_{ij} = 0$.

3.1.1 Input parameters

1. **'data_samples': array_like of 2D ndarray**
   Input array, of arrays with double.
   - A list of K matrices, where each entry of the list represents a distinct data object.
   - The i-th entry in the list is a matrix of size $n_i \times d$:
     - n_i represents the number of data points in the i-th data object.
     - d is the dimension of each data point.
2. **'base_norm_function_name':callable**. A callable function with 2 parameters. Each parameter represents array_like in d-dimension, and the output is a positive double that is the distance between those points. This function will be called for the distances on the base manifold. The default will be the Euclidean norm ('EuclideanNorm').

   Python Data type: Python function, with a signature

   base_norm_function_name(X, Y)

   #### 3.1.2 Input parameters

   (a) **X: array_like in d-dimension**

   (b) **Y: array_like in d-dimension**

3.1.3 Returns:

out: double positive array_like returns a list of all the distances between each pair of observations in $X \in \mathbb{R}^{N\times d}$ and $Y \in \mathbb{R}^{M\times d}$.

3.1.4 Raises:

EmptyDataError - if X or Y is empty.
NotSameDim - if the dimensions of X and Y do not match.

3. **'sparsity_param_base': Non-negative double**.

   Sparsity radius on the base manifold, which controls the sparsity of the base manifold. It is used to find the 'sparsity_param_base' -nearest neighbors within the specified radius.

4. **'data_object_maps': dictionary_like with 2D array_like** Mappings between the data object:

   - A dictionary where
     - keys are tuples (i, j) representing the indices of data objects
     - values are M_{ij} an $n_i \times n_j$ matrix indicating the mapping from object i to object j.
   - The mappings define the correct order of correspondence between two data objects. For example, this can be the output of Continuous Procrustes Distance

5. **'subsample_mapping': double between 0-1**

   Subsample percentage of the data for calculation, should the correspondence mapping be used as is or sub-sampled. The default value is 1, use all the data, and otherwise sample the specified percentage of the data.

3.1.5 Returns

'base_dist_mat' : 2D array_like

base diffusion matrix B ($K \times K$ matrix).

3.1.6 Raises:

NegativeDistError - if the value returned from the base_norm_function_name is non-positive.
InputNumOfDistParam - the number of parameters in base_norm_function_name doesn't match.
NonBoundedSubSample - subsample_mapping is out of the range [0, 1].
NegativeSparsity - the sparsity_param_base parameter is negative.

3.2 ComputeFiberDist

ComputeFiberDist(data_samples, fiber_norm_func_name = 'EuclideanNorm', sparsity_param_fiber = 20, subsample_mapping = 1, data_object_maps, base_dist_mat)

Construct a fiber diffusion sparse block matrix F, a $\sum n_i \times \sum n_i$ matrix, where n_i represents the number of data points in the i-th data object. And the i, j block is of size $n_i \times n_j$.

1. If 'subsample_mapping' is not 1, sample each 'data_samples' object with respect to the given 'subsample_mapping' percentage, and create a subsampled version of the 'data_samples'.

2. For each $i = 1, \ldots, K$, and for each for each $j = 1, \ldots, K$

(a) If the (i, j) entry in 'base_dist_mat' is 0, set the $n_i \times n_j$ block in 'fiber_dist_mat' to be zero block.

(b) Otherwise

(c) Let S_i (of size $n_i \times d$) and S_j (of size $n_j \times d$) denote the data matrix of object i and j. Let M_{ij} be the mappings from object i to j.

(d) For each $k = 1, \ldots, n_i$, and for each for each $t = 1, \ldots, n_j$, update the (k, t)th entry in 'fiber_dist_mat_inner' by

- Set 'fiber_dist_mat_inner'(k, t) by applying the 'fiber_norm_function_name' function to k, and t, corresponding points (using 'data_object_maps'), from S_i, and s_j objects. Then fiber_dist_mat_inner(k, t) = fiber_norm_function_name(data_object_maps $(i, j)(S_i, S_j)$).
- if 'fiber_dist_mat_inner'(k, t) is larger than the fiber sparsity radius ('fiber_dist_mat'), then set fiber_dist_mat_inner$(k, t) = 0$.

3. Set the $n_i \times n_j$ block in 'fiber_dist_mat' to the 'fiber_dist_mat_inner'

3.2.1 Input parameters

1. **'data_samples': array_like of 2D ndarray Input array, of arrays with double.**

   - A list of K matrices, where each entry of the list represents a distinct data object.
   - The i-th entry in the list is a matrix of size $n_i \times d$:
     - n_i represents the number of data points in the i-th data object.
     - d is the dimension of each data point.

   'fiber_norm_function_name':callable. A callable function with 2 parameters. Each parameter represents array_like in d-dimension, and the output is a positive double that is the distance between those points. This function will be called for the distances on the fiber. The default will be the Euclidean norm ('EuclideanNorm').

   Python Data type: Python function, with a signature

   fiber_norm_function_name(X, Y)

3.2.2 Input parameters

(a) **X: array_like in d-dimension**

(b) **Y: array_like in d-dimension**

3.2.3 Returns:

out: double positive array_like returns a list of all the distances between each pair of observations in $X \in \mathbb{R}^{N \times d}$ and $Y \in \mathbb{R}^{M \times d}$.

3.2.4 Raises:

EmptyDataError - if X or Y is empty.
NotSameDim - if the dimensions of X and Y do not match.

2. **'sparsity_param_fiber': Non-negative double**.

   Sparsity radius on the fiber manifold, which controls the sparsity of the fiber. It is used to find the 'sparsity_param_fiber' -nearest neighbors within the specified radius.

3. **'data_object_maps': dictionary_like with 2D array_like** Mappings between the data object:

   - A dictionary where
     - keys are tuples (i, j) representing the indices of data objects

- values are M_{ij} an $n_i \times n_j$ matrix indicating the mapping from object i to object j.
- The mappings define the correct order of correspondence between two data objects. For example, this can be the output of Continuous Procrustes Distance

4. **'subsample_mapping': double between 0-1**

   Subsample percentage of the data for calculation, should the correspondence mapping be used as is or sub-sampled. The default value is 1, use all the data, and otherwise sample the specified percentage of the data.

   'base_dist_mat' : 2D array_like

   base diffusion matrix B ($K \times K$ matrix).

3.2.5 Returns

'fiber_dist_mat' - 2D sparse array like. A $\sum n_i \times \sum n_i$ block distance matrix on the fiber, n_i represents the number of data points in the i-th data object.

3.2.6 Raises:

[Shan todo add more raise everywhere.]

3.3 ComputeHorizontalDiffusionLaplacian

ComputeHorizontalDiffusionLaplacian(base_dist_mat, fiber_dist_mat, kernelFunc_base, kernel_func_fiber, diffusion_param_base, diffusion_param_fiber)

Perform the HDM calculation for K objects, with each object is in $\mathbb{R}^{n_i \times d}$, where $i = 1, \ldots K$. For example, a typical scenario is comparing 100 shapes, with 5000 points each in 3 dimensions, the HDM matrix 'H', see below will be of size 100×5000.

Below is a pseudocode. Construct the horizontal diffusion matrix H ($K \times K$ block matrix)

- Each block H_{ij} is a matrix of size $n_i \times n_j$.
- Construct the base diffusion matrix ('base_diffusion_matrix') by applying the base kernel function ('kernel_func_base') and base diffusion parameter ('diffusion_param_base') on the base_dist_mat
- Construct the fiber diffusion matrix ('fiber_diffusion_matrix') by applying the fiber kernel function ('kernel_func_fiber') and fiber diffusion parameter ('diffusion_param_fiber') on the fiber_dist_mat
- Each entry in the block $H_{ij}^{s,t}$ is obtained using the following steps:
  - For each block (i, j) multiply the base_diffusion_matrix(i, j entry with the (i, j) block from the fiber_diffusion_matrix
    all entries in the block with B_{ij}
- Once the block matrix is constructed, we create a symmetric version my $H = \frac{1}{2}(H + H')$, and normalize it so that each row/ column will sum up to 1.

3.3.1 Input parameters

1. **'base_dist_mat':** 2D array like. A $K \times K$ distance matrix on the base manifold.
2. **'fiber_dist_mat':** 2D sparse array like. A $K \times K$ block matrix representing distance on the fiber.
3. **'kernel_func_base':** Callable function that takes in one distance value and outputs the similarity score. The default will be Gaussian.

   add more details about the kernel function.

   Python Data type: Python function, with a signature

   kernel_func_base(D, diffusion_param_base) - given a list of distances, and 'diffusion_param_base', the callable function returns a list of values once applying the kernel on each distance, with respect to the diffusion parameter. Distances should be non-negative.
4. **'kernel_func_fiber':** Callable function that takes in one distance value and outputs the similarity score. The default will be Gaussian.

   Python Data type: Python function, with a signature

   kernel_func_fiber(D, diffusion_param_fiber) - given a list of distances, and 'diffusion_param_fiber', the callable function returns a list of values once applying the kernel on each distance, with respect to the diffusion parameter. Distances should be non-negative.
5. **'diffusion_param_base':** Base diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the base manifold. This is the parameter that goes into Base kernel function. For example, the base kernel function is $exp(-d^2/\epsilon)$, and this is the ϵ parameter
6. **'diffusion_param_fiber':** Fibre diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the fibers. Similar to Base diffusion parameter.

3.3.2 Returns

1. **horizontal_diffusion_laplacian** - (2D sparse array like) the horizontal diffusion $K \times K$ block distance matrix of size $\sum n_i \times \sum n_i$.

3.4 eigenDecomposision

eigenDecomposision(horizontal_diffusion_laplacian, num_return_eig_vec)

Calculate the eigendecomposition and output the top 'num_return_eigen_vec' eigenvalues and eigenvectors found (ordered decreasingly by the eigenvalues).

The eigenvectors Ψ_r of H have $n_1 + n_2 + \ldots + n_K$ entries . We can consider the eigenvectors as concatenations of K segments of lengths N_k, each indexed according to surface S_k, with k ranging from 1 to K.

3.4.1 Input parameters

1. **horizontal_diffusion_laplacian** - (2D sparse array like) the horizontal diffusion $K \times K$ block distance matrix of size $\sum n_i \times \sum n_i$.
2. **'num_return_eigen_vec':**- positive integer - number of eigenvectors, and eigenvalues to return, default 4. The number will be in the range 1-30.

3.4.2 Returns

1. **'eigenvalues'** - list of eigenvalues in decreasing order.
2. **'eigenvectors'** - list of eigenvectors in the corresponding order to the eigenvalues in decreasing order.

3.5 kernelGaussianFunc

kernelGaussianFunc(D, param_list)

Kernel function that outputs the values of the kernel for the given distance list, and the given parameter

Notify the user in case of a failure in: The entries of D are negative

3.5.1 Input parameters

1. **D:** - a list of positive doubles, representing distances
2. **'param_list'** - non-negative double, which controls the amount of information that is taken into account by the kernel.

3.5.2 Returns

'kernel_vals'- a list of values once applying the kernel on each distance, with respect to the given parameter.

3.6 estimateDiffusionParamBase

estimateDiffusionParamBase(base_dist_mat, fiber_dist_mat, kernelFunc_base, kernel_func_fiber, diffusion_param_base, diffusion_param_fiber)

Find the diffusion parameter, where the minimum of the semi-group property is reached :

$$SGE(t) = ||H_t^2 - H_{2t}||$$

3.6.1 Input parameters

1. **'base_dist_mat':** Distance matrix on the base manifold
   - A $K \times K$ matrix, where each entry d_{ij} indicates the distance between data object i and data object j.
   - If this is None, we compute the distance matrix according to base distance metric 'base_norm_func_name'.
2. **'fiber_dist_mat':** Distance matrix on the fiber
   - A $K \times K$ block matrix, which corresponds to H in the pseudocode and is provided by the user.
   - If this is None, we compute H using the HDM algorithm

3. **'kernel_func_base':** Callable function that takes in one distance value and outputs the similarity score. The default will be Gaussian.

   Python Data type: Python function, with a signature

   kernel_func_base(D, diffusion_param_base) - given a list of distances, and 'diffusion_param_base', the callable function returns a list of values once applying the kernel on each distance, with respect to the diffusion parameter. Distances should be non-negative.

4. **'kernel_func_fiber'):** Callable function that takes in one distance value and outputs the similarity score. The default will be Gaussian.

   Python Data type: Python function, with a signature

   kernel_func_fiber(D, diffusion_param_fiber) - given a list of distances, and 'diffusion_param_fiber', the callable function returns a list of values once applying the kernel on each distance, with respect to the diffusion parameter. Distances should be non-negative.

5. **'diffusion_param_base':** Base diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the base manifold. This is the parameter that goes into Base kernel function. For example, the base kernel function is $exp(-d^2/\epsilon)$, and this is the ϵ parameter

6. **'diffusion_param_fiber':** Fibre diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the fibers. Similar to Base diffusion parameter.

3.6.2 Returns

'diffusion_param_base': Base diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the base manifold.

3.7 estimateDiffusionParamFiber

estimateDiffusionParamFiber(base_dist_mat, fiber_dist_mat, kernelFunc_base, kernel_func_fiber, diffusion_param_base, diffusion_param_fiber)

[Shira - review]

TBD

Notify the user in case of a failure in: The entries of D are negative

3.7.1 Input parameters

[Shira: review the Input parameters]

1. 'base_dist_mat': Distance matrix on the base manifold
   - A $K \times K$ matrix, where each entry d_{ij} indicates the distance between data object i and data object j.
   - If this is None, we compute the distance matrix according to base distance metric 'base_norm_func_name'.
2. 'fiber_dist_mat': Distance matrix on the fiber
   - A $K \times K$ block matrix, which corresponds to H in the pseudocode and is provided by the user.
   - If this is None, we compute H using the HDM algorithm

3. Base kernel function ('kernel_func_base'): Callable function that takes in one distance value and outputs the similarity score. The default will be Gaussian.

   Python Data type: Python function, with a signature

   kernel_func_base(D, diffusion_param_base) - given a list of distances, and 'diffusion_param_base', the callable function returns a list of values once applying the kernel on each distance, with respect to the diffusion parameter. Distances should be non-negative.

4. Fiber kernel function ('kernel_func_fiber'): Callable function that takes in one distance value and outputs the similarity score. The default will be Gaussian.

   Python Data type: Python function, with a signature

   kernel_func_fiber(D, diffusion_param_fiber) - given a list of distances, and 'diffusion_param_fiber', the callable function returns a list of values once applying the kernel on each distance, with respect to the diffusion parameter. Distances should be non-negative.

5. 'diffusion_param_base': Base diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the base manifold. This is the parameter that goes into Base kernel function. For example, the base kernel function is $exp(-d^2/\epsilon)$, and this is the ϵ parameter

6. 'diffusion_param_fiber': Fibre diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the fibers. Similar to Base diffusion parameter.

3.7.2 Returns

'diffusion_param_fiber': Fiber diffusion parameter. non-negative double, which controls the amount of information that is taken into account from the fiber manifold.