Appendix B

Settings File: Parameter Names and Values

Note that optional settings can either be omitted entirely from the settings file, or can have only their values omitted. The spelling of setting names and values in the settings file is case-independent.

Note that for any settings parameter that accepts a set of string values, and that has an associated default value, the string "default" can be specified in the settings file. The output by the application will record the actual value that was being used for program execution, prefixed by "defaulted to".

Setting Name	**Allowed Values**	**Explanation**
Project information		
project	Any string	Provided to users for organizing their data. No restriction on user choice
dataset	Any string	Provided to users for organizing their data. No restriction on user choice
user	Any string	Provided to users for organizing their data. No restriction on user choice
notes	Any string	Provided to users for organizing their data. No restriction on user choice
Application mode selections		
applicationMode	Valid string specifying the mode of the application • api • standalone	The default mode is standalone mode, which means that all results from the application are saved to files (i.e., 0- and 1-dimensional intervals, general runtime results, etc). In api mode, the output to files, as well as to the command line, is

		turned off. Instead, the user would use the return values from various methods to obtain the results computed by the application.
feedbackLevel	[To be implemented] Valid string specifying the level of feedback provided by the application: • verbose • basic • none	The feedbackLevel provides for overriding the default feedback from the application: in api mode, the default output is basic, whereas in standalone it is verbose. However, depending on the personal preferences of the user, or the way the application is being run, e.g., in batch mode on a cluster, these settings can be changed.
suppressAllOutput	Does what it says, namely omit ALL output from the application. Valid values are • yes • no	Defaults to 'no'. Was introduced for use in the API mode, so any 'front end' code can get the output by calling back the tda object, while all output to the command line is being suppressed.
Task component specifications		
taskChoice	Valid string specifying the task to be run: • LSD (local spherical distance) • M12 • M23 • RCA0 • M01 (to be implemented) • skip	
Selections for Algorithm 'M12', 'M23', and 'M01'		
distanceBoundOnEdges	Real number greater than 0	
supplyDataAs	Valid string from: • pointCloud • distanceMatrix • sparseMatrix • object (to be implemented)	When 'pointCloud' is specified, the input needs to be provided as a valid file containing the listing of the points. The code will then compute the edges using the specified metric, and create the internal edgeList. When 'distanceMatrix' is specified, the input file needs to specify the entire matrix of pair-wise distances between the points. Note that only

		values above the diagonal are being used (though the input needs to be a full matrix). Any value of 0 will be treated as "no edge" between the corresponding vertices. When 'sparseMatrix' is specified, the format needs to be in form of triples on each input line (i.e., each edge], as follows: 'vertexIndex1 vertexIndex2 distanceBetweenTheVertices'. Please note that each vertexIndex needs to be an integer. The order in which the vertexIndices are listed is not significant, though internally we always use the information in ascending order. When the 'object' option is selected, we compute the point cloud from the specified object. Currently, this is only used for internal testing.
pointCloudFile	File containing the supplied "point cloud" data, which can in fact be supplied as a point cloud, a distance matrix, a sparse matrix, or an object.	The content of the file needs to follow the format of the type of date (as specified via supplyDataAs), namely list of points, or distance matrix.
convertDataToSparseMatrixFormat	Valid string specifying whether to output the point cloud data in sparseMatrix format: • yes • no	This is a convenience option: when the input is in any of the other input formats (see 'supplyDataAs'), the application outputs the edges in sparse matrix format to a file. Note that the sparse matrix data that is being written to file is based on the list of edges **after the distance bound is applied**, i.e., it is **not** the entire set of points that was potentially provided via the original input! Hint: if one wants to have the "complete" sparse matrix of all points in the original set, one needs to specify a large distance bound that "catches" all the points.
sparseMatrixOutputFile	Valid string for specifying the file for writing the sparse matrix to.	
convertDataToDistanceMatrixFormat	Valid string specifying whether to	This is a convenience option: when

	output the point cloud data in distanceMatrix format: • yes • no	the input is in any of the other input formats (see ‘supplyDataAs’), the application outputs the edges in distance matrix format to a file.
distanceMatrixOutputFile	Valid string for specifying the file for writing the distance matrix to.	
0DintervalsFile	Any valid file name; • May be specified using a time stamp token as part of the name.	The name of the file for the computed 0-dimensional intervals (as computed by the M12 algorithm).
1DintervalsFile	Any valid file name; • May be specified using a time stamp token as part of the name.	The name of the file for the computed 1-dimensional intervals (as computed by the M12 algorithm).
1DintervalsAndRepsFile	Any valid file name; • May be specified using a time stamp token as part of the name.	The name of the file for the computed intervals and representatives (as computed by the M01 algorithm).
Input and output locations		
inputDirectory	Any valid directory path; can be specified as (valid) absolute or relative path	Specifies the directory where the input files for the current data run are located.
outputDirectory	Any valid directory path; can be specified as (valid) absolute or relative path	Specifies the directory where the output files for the current data run will be placed.
reportFile	Any valid file name; • May be specified using a time stamp token as part of the name.	The name of the (detailed) report file.
intervalsFile	Any valid file name; • May be specified using a time stamp token as part of the name.	The name of the file for the computed intervals (M12 algorithm).
intervalsAndGeneratorsFile	Any valid file name; • May be specified using a time stamp token as part of the name.	The name of the file for the computed intervals and generators (M01 algorithm).

Used Metric		
metricChoice	Valid string choices are: • L1 • L2 • Lp • Linf If not specified, defaults to using the L2 metric.	L1, L2, L-p and L-inf metrics are implemented.
p_valueForLpMetric	Integer value greater than 0, when L-p metric is selected as the metric choice.	
Z/Z_p computation		
zp_value	Integer greater than 0. When not specified, defaults to 2 (i.e. computations are done Z/Z_2)	Specifies the base p of the Z_p-based computations.
Performance settings		
switchToSparseAt	Integer greater than 0.	Specifies the switch-over point between using internal data representations of ‘distance matrix’ (below the threshold) and ‘sparse matrix’ (above the threshold)

Appendix C

API Calls

The methods exposed as part of the TDA API are located in the *tda* class, in the *tda.api* package, of the jar file.

Creating a Tda Object		
Tda()		Simplest constructor, no options specified for initialization. Note that any new Tda object will start from a fresh slate (i.e., the *reset()* method will be applied as part of the internal code)
Tda(String)	String contains a single "setting=value" pair as argument.	The supplied *value* will be applied to the *setting.*
Tda(String[])	Argument is an array of "setting=value" pairs.	Each specified setting is being set to its supplied value.
General Methods		
reset()		Resets the internal properties of the Tda object.
applyData(String)	String contains a single "setting=value" pair as argument.	Applies the supplied *value* to the *setting.*
applyData(String[])	Argument is an array of "setting=value" pairs.	Each specified setting is being set to its supplied value.
applyData(double[][])	Loading an array of data (use is determined by the algorithm – generally it is the set of points that make up the point cloud)	
applyData(double[])	Loading an array of data (use is determined by the algorithm –	

	generally it is the set of points that make up the point cloud) This is a special case of the 2-dimensional array, provided for environments like Matlab.	
applyData(String[], double[][])	Combines the previous 2 calls (for convenience, and for preserving legacy code)	
getResults()		Get a listing of the results that the Tda has available after executing. The listing depends on the algorithm.
getResults(int)		Return the specific result as specified by *int*.
getData()		Get the set of assigned Data for the Tda object.
getParameters()		Get the entire set of parameters used by the Tda object.
Algorithms		
RCA0()		Computes the 0-dimensional persistent homology.
RCA0(String[])	Argument is an array of “setting=value” pairs.	Computes RCA0, with the additional argument(s).
RCA0(double[])	2-dimensional array specifies the coordinates for the points in the point cloud.	Computes RCA0, with the additional argument.
RCA0(String[], double[])	Combines the previous 2 calls (for convenience, and for preserving legacy code)	Computes RCA0, with the additional arguments.
RCA1()		Computes the 1-dimensional persistent homology.
RCA1(String[])	Argument is an array of “setting=value” pairs.	Computes RCA1, with the additional argument(s).
RCA1(double[][])	2-dimensional array specifies the coordinates for the points in the point cloud.	Computes RCA1, with the additional argument.
RCA1(String[], double[])	Combines the previous 2 calls (for convenience, and for preserving legacy code)	Computes RCA1, with the additional arguments.

RCA2()		Computes the 2-dimensional persistent homology.
RCA2(String[])	Argument is an array of "setting=value" pairs.	Computes RCA2, with the additional argument(s).
RCA2(double[])	2-dimensional array specifies the coordinates for the points in the point cloud.	Computes RCA2, with the additional argument.
RCA2(String[], double[])	Combines the previous 2 calls (for convenience, and for preserving legacy code)	Computes RCA2, with the additional arguments.
LSD()		Computes the least spherical distance matrix.
LSD(double[][], double, double[])	Arguments are the point cloud, distance, and center point. In n-dimensions.	Executes LSD with the specified data.
LSD(double[], double, double)	Arguments are the point cloud, distance, and center point. In 1 dimension – this is obviously a special case of the n-dimensional data, but needed for environments like Matlab.	Executes LSD with the specified data.

API Settings

The following settings are useful when accessing TDA via its API calls.

applicationMode	Select value 'api'.	The 'api' value will restrict the output from TDA to a bare minimum. In particular, no output to file will be attempted.
suppressAllOutput	Optional value, set to 'yes'.	When set to 'yes', all (command line) output from TDA is omitted.

Appendix D

References

For more information about TDA and its background, please refer to John Harer and Paul Bendich's web pages at the Duke Mathematics Department:

http://fds.duke.edu/db/aas/math/faculty/john.harer
http://fds.duke.edu/db/aas/math/faculty/bendich