
ElasticFDA.jl Documentation

Release 0.2.1

J. Derek Tucker

Oct 28, 2018

Contents

1	Getting Started	3
1.1	Installation	3
1.2	References	3
2	Functional Alignment	5
2.1	SRSF Functions	5
2.2	Alignment	6
3	Functional Principal Component Analysis	9
3.1	fPCA Functions	9
4	Bayesian Alignment	11
4.1	Alignment	11
5	Elastic Functional Regression	13
5.1	Regression Models and Prediction	13
6	Curve Alignment	17
6.1	SRVF Functions	17
6.2	Alignment and Statistics	18
7	Image Alignment	21
7.1	Alignment	21

The *ElasticFDA* package provides a collection of functions for functional data analysis using the square-root slope framework and curves using the square-root velocity framework which performs pair-wise and group-wise alignment as well as modeling using functional component analysis and regression.

Contents:

1.1 Installation

The ElasticFDA package is available through the Julia package system by running `Pkg.add("ElasticFDA")`. Throughout, we have assume that you have installed the package.

This package relies on two *c/cpp* optimization routines which will either compile with `icc` or `g++`. One of the libraries relies LAPACK and BLAS. The makefile will detect if `icc` is installed and use it, otherwise it will default to `g++`. If `icc` is detected it will use MKL as the BLAS and LAPACK implementation. Otherwise OpenBLAS is used/required.

1.2 References

This package is based on code from the following publications:

- Tucker, J. D. 2014, Functional Component Analysis and Regression using Elastic Methods. Ph.D. Thesis, Florida State University.
- Robinson, D. T. 2012, Function Data Analysis and Partial Shape Matching in the Square Root Velocity Framework. Ph.D. Thesis, Florida State University.
- Huang, W. 2014, Optimization Algorithms on Riemannian Manifolds with Applications. Ph.D. Thesis, Florida State University.
- Srivastava, A., Wu, W., Kurtek, S., Klassen, E. and Marron, J. S. (2011). Registration of Functional Data Using Fisher-Rao Metric. [arXiv:1103.3817v2 \[math.ST\]](https://arxiv.org/abs/1103.3817v2).
- Tucker, J. D., Wu, W. and Srivastava, A. (2013). Generative models for functional data using phase and amplitude separation. *Computational Statistics and Data Analysis* 61, 50-66.
- Tucker, J.D., Wu, W. and Srivastava, A. (2014). Phase-Amplitude Separation of Proteomics Data Using Extended Fisher-Rao Metric. *Electronic Journal of Statistics* 8 (2), 1724-1733.
- Tucker, J.D., Wu, W. and Srivastava, A. (2014). Analysis of signals under compositional noise With applications to SONAR data. *IEEE Journal of Oceanic Engineering* 29 (2), 318-330.

- Kurtek, S., Srivastava, A. and Wu, W. (2011). Signal estimation under random time-warpings and nonlinear signal alignment. In Proceedings of Neural Information Processing Systems (NIPS).
- Joshi, S.H., Srivastava, A., Klassen, E. and Jermyn, I. (2007). A Novel Representation for Computing Geodesics Between n-Dimensional Elastic Curves. IEEE Conference on computer Vision and Pattern Recognition (CVPR), Minneapolis, MN.
- Srivastava, A., Klassen, E., Joshi, S., Jermyn, I., (2011). Shape analysis of elastic curves in euclidean spaces. Pattern Analysis and Machine Intelligence, IEEE Transactions on 33 (7), 1415-1428.
- Wen Huang, Kyle A. Gallivan, Anuj Srivastava, Pierre-Antoine Absil. “Riemannian Optimization for Elastic Shape Analysis”, Short version, The 21st International Symposium on Mathematical Theory of Networks and Systems (MTNS 2014).
- 17. Xie, S. Kurtek, E. Klassen, G. E. Christensen and A. Srivastava. Metric-based pairwise and multiple image registration. IEEE European Conference on Computer Vision (ECCV), September, 2014
- Cheng, W., Dryden, I. L., & Huang, X. (2016). Bayesian registration of functions and curves. Bayesian Analysis, 11(2), 447–475.

Functional Alignment

The main functions deal with the alignment of functional data using the *square-root slope (srsf)* framework. Where an input into a function is expecting an array of functions the shape is assumed to be (M, N) with M being the number of sample points and N being the number of functions.

2.1 SRSF Functions

f_to_srsf ($f::Array$, $timet=0$, $smooth=false$)

Convert function to square-root slope (srsf) functions

f is an array of shape (M, N) as described above. By default the function will generate timing information, otherwise $timet$ should be vector of length M describing the timing information. If $smooth=true$ the input data will be smoothed first using smoothing splines.

srsf_to_f ($q::Array$, $timet$, $f0=0.0$)

Convert srsf to function space

q is an array with the standard shape. $timet$ is a vector of timing information. $f0$ is the initial value of the function in f -space, this is required to make the transformation a bijection.

smooth_data ($f::Array$, $sparam=10$)

Smooth functional data using a box filter

f is an array with the standard shape. $sparam$ is the number of times to run the filter.

smooth_data! ($f::Array$, $sparam=10$)

same as `smooth_data`, except the smoothing is done in-place

trapz ($x::Vector$, $y::Array$, $dim=1$)

Trapezoidal Integration

x is a vector of time samples. y is the response and dim is the dimension to integrate along.

optimum_reparam ($q1$, $timet$, $q2$, $lam=0.0$, $method="DP"$, $w=0.01$, $f1o=0.0$, $f2o=0.0$)

Calculates the optimum reparametrization (warping) between two srsfs $q1$ and $q2$.

q_1 and q_2 can be vectors or arrays of the standard shape. $timet$ is a vector describing the time samples. lam controls the amount of warping. $method$ is the optimization method to find the warping. The default is Simultaneous Alignment (“SIMUL”). Other options are Dynamic Programming (“DP” or “DP2”) and Riemannian BFGS (“RBFSG”).

warp_f_gamma ($time::Vector, f::Vector, gam::Vector$)

Warp function f by warping function $gamma$

warp_q_gamma ($time::Vector, q::Vector, gam::Vector$)

Warp srsf q by warping function $gamma$

elastic_distance ($f1::Vector, f2::Vector, timet::Vector, method="SIMUL"$)

Calculates the elastic distance between two functions and returns the amplitude distance da and phase distance dp .

rgam ($N, sigma, num$)

Generate random warping functions of length N . $sigma$ controls the standard deviation across the random samples and num is the number of random samples.

2.2 Alignment

srsf_align ($f, timet; method="mean", smooth=false, sparam=10, lam=0.0, optim="DP", MaxItr=20$)

Aligns a collection of functions using the elastic square-root slope (srsf) framework.

- f is an array of shape (M,N) of N functions with M samples
- $timet$ is a vector of size M describing the sample points
- $method$ (string) calculate Karcher Mean or Median (options = “mean” or “median”) (default=“mean”)
- $smooth$ Smooth the data using a box filter (default = false)
- $sparam$ Number of times to run smoothing filter (default 10)
- lam controls the elasticity (default = 0)
- $optim$ optimization method to find warping, default is Simultaneous Alignment (“SIMUL”). Other options are Dynamic Programming (“DP2”), Riemannian BFGS (“RBFSG”)
- $MaxItr$ maximum number of iterations

Returns Dict containing:

- fn aligned functions - array of shape (M,N) of N functions with M samples
- qn aligned srsfs - similar structure to fn
- $q0$ original srsfs - similar structure to fn
- $fmean$ function mean or median - vector of length N
- mqn srsf mean or median - vector of length N
- gam warping functions - similar structure to fn
- $orig_var$ Original Variance of Functions
- amp_var Amplitude Variance
- $phase_var$ Phase Variance

align_fPCA ($f, timet; num_comp=3, smooth=false, sparam=10, MaxItr=50$)

Aligns a collection of functions while extracting principal components. The functions are aligned to the principal components

- `f` array of shape (M,N) of N functions with M samples
- `timet` vector of size M describing the sample points
- `num_comp` Number of components (default = 3)
- `smooth` Smooth the data using a box filter (default = false)
- `sparam` Number of times to run smoothing filter (default 10)
- `MaxIter` maximum number of iterations

Returns Dict containing:

- `fn` aligned functions - array of shape (M,N) of N functions with M samples
- `qn` aligned srvfs - similar structure to `fn`
- `q0` original srvf - similar structure to `fn`
- `mqn` srvf mean or median - vector of length M
- `gam` warping functions - similar structure to `fn`
- `q_pca` srsf principal directions
- `f_pca` functional principal directions
- `latent` latent values
- `coef` coefficients
- `U` eigenvectors
- `orig_var` Original Variance of Functions
- `amp_var` Amplitude Variance
- `phase_var` Phase Variance

Functional Principal Component Analysis

These functions are for computing functional principal component analysis (fPCA) on aligned data and generating random samples

3.1 fPCA Functions

vert_fPCA (*fn, timet, qn; no=1*)

Calculates vertical functional principal component analysis on aligned data

- *fn* array of shape (M,N) of N aligned functions with M samples
- *timet* vector of size M describing the sample points
- *qn* array of shape (M,N) of N aligned SRSF with M samples
- *no* number of components to extract (default = 1)

Returns Dict containing:

- *q_pca* srsf principal directions
- *f_pca* functional principal directions
- *latent* latent values
- *coef* coefficients
- *U* eigenvectors

horiz_fPCA (*gam, timet; no=1*)

Calculates horizontal functional principal component analysis on aligned data

- *gam* array of shape (M,N) of N warping functions with M samples
- *timet* vector of size M describing the sample points
- *no* number of components to extract (default = 1)

Returns Dict containing:

- `gam_pca` warping principal directions
- `psi_pca` srsf functional principal directions
- `latent` latent values
- `U` eigenvectors
- `gam_mu` mean warping function
- `vec1` shooting vectors

gauss_model (*fn, timet, qn, gam; n=1, sort_samples=false*)

Computes random samples of functions from aligned data using Gaussian model

- `fn` aligned functions (M,N)
- `timet` vector (M) describing time
- `qn` aligned srvfs (M,N)
- `gam` warping functions (M,N)
- `n` number of samples
- `sort_samples` sort samples

Returns Dict containing:

- `fs` random aligned functions
- `gams` random warping functions
- `ft` random functions

The following functions align functional data in the srsf framework using a Bayesian approach. These functions are *experimental* and results are not fully tested

4.1 Alignment

pair_warping_baye (*f1, f2; iter=15000, times=5, powera=1*)

Compute pair warping between two functions using Bayesian method

- *f1, f2* vectors describing functions
- *iter* number of iterations
- *times* MCMC parameter
- *powera* MCMC parameter

Returns Dict containing:

- *f1* function *f1*,
- *f2_q* srsf registration,
- *gam_q* warping function,
- *f2a* registered *f2*,
- *gam* warping function,
- *dist_collect* distance,
- *best_match* best match,

group_warping_bayes (*f; iter=20000, times=5, powera=1*)

Group alignment of functions using Bayesian method

- *f* array (M,N) of N functions,
- *iter* number of MCMC iterations,

- `times` time slicing,
- `powera` MCMC parameter,

Returns Dict containing:

- `f_q` registered srvfs
- `gam_q` warping functions
- `f_a` registered functions
- `gam_a` warping functions

Elastic Functional Regression

These functions compute elastic standard, logistic, and m-logistic regression models. This code is experimental and results are not guaranteed

5.1 Regression Models and Prediction

elastic_regression (*f*, *y*, *timet*; *B=None*, *lambda=0*, *df=20*, *max_itr=20*, *smooth=false*)

Calculate elastic regression from function data *f*, for response *y*

- *f* array (M,N) of N functions
- *y* vector (N) of responses
- *timet* vector (N) describing time samples
- *B* matrix describing basis functions (M,N) (default=None generates a B-spline basis)
- *lambda* regularization parameter
- *df* degree of freedom of basis
- *max_itr* maximum number of iterations
- *smooth* smooth data

Returns Dict describing regression:

- *alpha* intercept
- *beta* regression function
- *fn* aligned functions
- *qn* aligned srsfs
- *gamma* warping functions
- *q* original srsfs

- B basis functions
- `type` type of regression
- `b` coefficients
- SSE sum of squared error

elastic_logistic (*f*, *y*, *timet*; *B=None*, *df=20*, *max_itr=20*, *smooth=false*)

Calculate elastic logistic regression from function data *f*, for response *y*

- *f* array (M,N) of N functions
- *y* vector (N) of responses
- *timet* vector (N) describing time samples
- *B* matrix describing basis functions (M,N) (default=None generates a B-spline basis)
- *df* degree of freedom of basis
- *max_itr* maximum number of iterations
- *smooth* smooth data

Returns Dict describing regression:

- `alpha` intercept
- `beta` regression function
- `fn` aligned functions
- `qn` aligned srsfs
- `gamma` warping functions
- `q` original srsfs
- *B* basis functions
- `type` type of regression
- `b` coefficients
- `LL` logistic loss

elastic_mlogistic (*f*, *y*, *timet*; *B=None*, *df=20*, *max_itr=20*, *smooth=false*)

Calculate elastic m-logistic regression from function data *f*, for response *y*

- “*f*: array (M,N) of N functions
- “*y*: vector (N) of responses
- “*timet*: vector (N) describing time samples
- “*B*: matrix describing basis functions (M,N) (default=None generates a B-spline basis)
- “*df*: degree of freedom of basis
- “*max_itr*: maximum number of iterations
- “*smooth*: smooth data

Returns Dict describing regression:

- `alpha` intercept
- `beta` regression function
- `fn` aligned functions

- `qn` aligned srsfs
- `gamma` warping functions
- `q` original srsfs
- `B` basis functions
- `type` type of regression
- `b` coefficients
- `n_classes` number of classes
- `LL` logistic loss

elastic_prediction (*f*, *timet*, *model*; *y=None*, *smooth=false*)

Prediction from elastic regression model

- `f` functions to predict
- `timet` vector describing time samples
- `model` calculated model (regression, logistic, mlogistic)
- `y` true responses (default = None)
- `smooth` smooth data (default = false)

Returns:

- `y_pred` predicted value
- `y_labels` labels of predicted value
- `Perf` Performance metric if truth is supplied

These functions are for processing of N-D curves using the *square-root velocity framework (srvf)*

6.1 SRVF Functions

curve_to_q(*beta*)

Convert curve to square-root velocity function (srvf)

beta is an array of shape (n,T) describing the curve, where n is the dimension and T is the number of sample points

q_to_curve(*q*)

Convert srvf to curve

q is an array of shape (n,T) describing the srvf, where n is the dimension and T is the number of sample points

optimum_reparam(*beta1, beta2, lam, method="DP", w=0.01, rotated=true, isclosed=false*)

Calculates the optimum reparamertization (warping) between two curves *beta1* and *beta2*, using the srvf framework

- *beta1* array (n,T) describing curve 1
- *beta2* array (n,T) describing curve 2
- *lam* control amount of warping (default=0.0)
- *method* optimization method to find warping, default is Dynamic Programming (“DP”). Other options are Coordinate Descent (“DP2”), Riemanain BFGS (“LRBFGS”).
- *w* Controls LRBFGS (default = 0.01)
- *rotated* calculate rotation (default = true)
- *isclosed* closed curve (default = false)

Returns:

- *gam* warping function

- R rotation matrix
- tau seed value

calc_shape_dist (*beta1*, *beta2*)

Calculate elastic shape distance between two curves *beta1* and *beta2*

beta1 and *beta2* are arrays of shape (n,T) describing the curve, where n is the dimension and T is the number of sample points

resamplecurve (*x*, *N=100*)

Resamples Curve

- *x* array describing curve (n,T)
- *N* Number of samples to re-sample curve, *N* usually is > T

6.2 Alignment and Statistics

curve_pair_align (*beta1::Array{Float64, 2}*, *beta2::Array{Float64, 2}*)

Pairwise align two curves

- *beta1* array (n,T)
- *beta2* array (n,T)

Returns:

- *beta2n* aligned curve 2 to 1
- *q2n* aligned srvf 2 to 1
- *gam* warping function
- *q1* srvf of curve 1

curve_geodesic (*beta1::Array{Float64, 2}*, *beta2::Array{Float64, 2}*, *k::Integer=5*)

Find curves along geodesic between two curves

- *beta1* array (n,T)
- *beta2* array (n,T)
- *k* number of curves along geodesic

Returns:

- *geod* curves along geodesic (n,T,k)
- *geod_q* srvf's along geodesic

curve_srvf_align (*beta*; *mode='O'*, *maxit=20*)

Aligns a collection of curves using the elastic square-root velocity (srvf) framework.

- *beta* array (n,T,N) for N number of curves
- *mode* Open ('O') or Closed ('C') curves
- *maxit* maximum number of iterations

Returns:

- *betan* aligned curves
- *qn* aligned srvfs

- `betamean` mean curve
- `q_mu` mean srvf

curve_karcher_mean (*beta; mode='O', maxit=20*)

Calculates Karcher mean of a collection of curves using the elastic square-root velocity (srvf) framework.

- `beta` array (n,T,N) for N number of curves
- `mode` Open ('O') or Closed ('C') curves
- `maxit` maximum number of iterations

Returns:

- `mu` mean srvf
- `betamean` mean curve
- `v` shooting vectors
- `q` array of srvfs

curve_karcher_cov (*betamean, beta; mode='O'*)

Calculate Karcher Covariance of a set of curves

- `betamean` array (n,T) of mean curve
- `beta` array (n,T,N) for N number of curves
- `mode` Open ('O') or Closed ('C') curves

Returns:

- `K` covariance matrix

curve_principal_directions (*betamean, mu, K; mode='O', no=3, N=5*)

Calculate principal directions of a set of curves

- `betamean` array (n,T) of mean curve
- `mu` array (n,T) of mean srvf
- `K` array (T,T) covariance matrix
- `mode` Open ('O') or Closed ('C') curve
- `no` number of components
- `N` number of samples on each side of mean

Returns:

- `pd` array describing principal directions

sample_shapes (*mu, K; mode='O', no=3, numSamp=10*)

Sample shapes from model

- `mu` array (n,T) mean srvf
- `K` array (T,T) covariance matrix
- `mode` Open ('O') or Closed ('C') curves
- `no` number of principal components
- `numSamp` number of samples

Return:

- `samples` array (n,T,numSamp) of sample curves

These functions are for processing of images using the *q-map framework*

7.1 Alignment

pair_align_image (*I1, I2; M=5, ortho=true, basis_type="t", resize=true, N=64, stepsize=1e-5, itermax=1000*)

Pairwise align two images

- *I1* reference image
- *I2* image to warp
- *M* number of basis elements
- *ortho* orthonormalize basis
- *basis_type* type of basis ("t", "s", "i", "o")
- *resize* resize image
- *N* size of resized image
- *stepsize* gradient stepsize
- *itermax* maximum number of iterations

Returns:

- *I2_new* aligned *I2*
- *gam* warping function

A

align_fPCA() (built-in function), 6

C

calc_shape_dist() (built-in function), 18
curve_geodesic() (built-in function), 18
curve_karcher_cov() (built-in function), 19
curve_karcher_mean() (built-in function), 19
curve_pair_align() (built-in function), 18
curve_principal_directions() (built-in function), 19
curve_srvf_align() (built-in function), 18
curve_to_q() (built-in function), 17

E

elastic_distance() (built-in function), 6
elastic_logistic() (built-in function), 14
elastic_mlogistic() (built-in function), 14
elastic_prediction() (built-in function), 15
elastic_regression() (built-in function), 13

F

f_to_srsf() (built-in function), 5

G

gauss_model() (built-in function), 10
group_warping_bayes() (built-in function), 11

H

horiz_fPCA() (built-in function), 9

O

optimum_reparam() (built-in function), 5, 17

P

pair_align_image() (built-in function), 21
pair_warping_bayes() (built-in function), 11

Q

q_to_curve() (built-in function), 17

R

resamplecurve() (built-in function), 18
rgam() (built-in function), 6

S

sample_shapes() (built-in function), 19
smooth_data() (built-in function), 5
srsf_align() (built-in function), 6
srsf_to_f() (built-in function), 5

T

trapz() (built-in function), 5

V

vert_fPCA() (built-in function), 9

W

warp_f_gamma() (built-in function), 6
warp_q_gamma() (built-in function), 6