

tidyfun: Tidy Functional Data

A new framework for representing and working with
function-valued data in R

Fabian Scheipl¹ Jeff Goldsmith²

¹: Dept. of Statistics, LMU Munich

²: Columbia University Mailman School of Public Health

JSM 2018

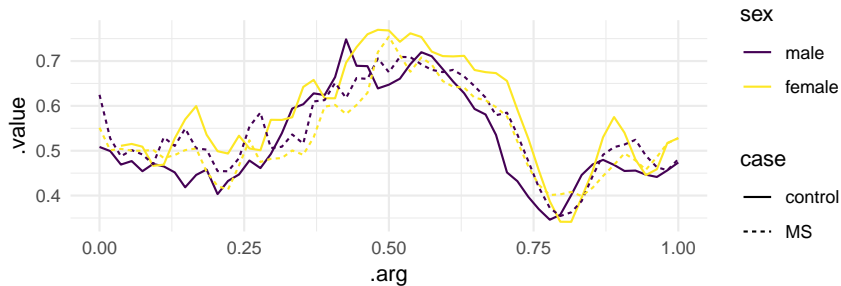


Let's start at the end...

This is what we're aiming for:

```
# group-wise functional medians:
```

```
medians = dti %>% group_by(case, sex) %>% summarize(median_rcst = median(rcst))  
ggplot(medians) + geom_spaghetti(aes(tf = median_rcst, col = sex, linetype = case))
```



```
glimpse(dti)
```

```
## Observations: 382
```

```
## Variables: 5
```

```
## $ id    <dbl> 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 101...
```

```
## $ sex   <fct> female, female, male, male, male, male, male, male, male,...
```

```
## $ case  <fct> control, control, control, control, control, control, control, con...
```

```
## $ cca   <tfd> 1001_1: (0.000,0.49);(0.011,0.52);(0.022,0.54); ..., 1002...
```

```
## $ rcst  <tfd> 1001_1: (0.000,0.26);(0.018,0.45);(0.037,0.40); ..., 1002...
```

tidyfun

The goal of **tidyfun** is to provide accessible and well-documented software that **makes functional data analysis in R easy**, specifically: data wrangling and exploratory analysis.

tidyfun provides:

- ▶ new **data types** for representing functional data: **tfd** & **tfb**
- ▶ arithmetic **operators**, descriptive **statistics** and **graphics** functions for such data
- ▶ tidyverse-verbs for handling functional data **inside** data frames.

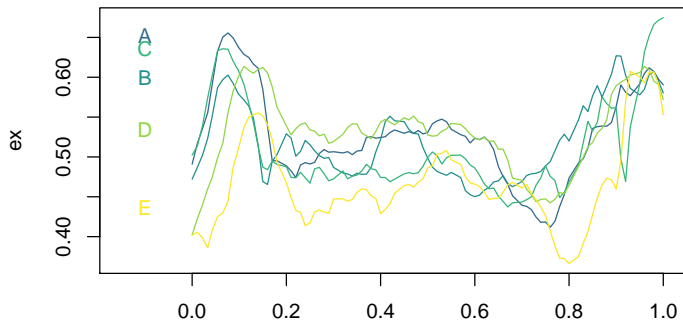
tf-Class: Definition

tf-class

`tf` is a new data type for (vectors of) functional data:

- ▶ an abstract superclass for functional data in 2 forms:
 - ▶ as (argument, value)-tuples: subclass `tfid`, also irregular or sparse
 - ▶ or in basis representation: subclass `tfb`
- ▶ basically, a `list` of numeric vectors
(... since `lists` work well as columns of data frames ...)
- ▶ with additional attributes that define *function-like* behavior:
 - ▶ how to **evaluate** the given “functions” for new arguments
 - ▶ their **domain**
 - ▶ the **resolution** of the argument values
- ▶ S3 based

Example Data



```
ex

## tfd[5] on (0,1) based on 93 evaluations each
## interpolation by approx_linear
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...
## C: (0.000,0.50);(0.011,0.51);(0.022,0.54); ...
## D: (0.000,0.40);(0.011,0.42);(0.022,0.44); ...
## E: (0.000,0.40);(0.011,0.41);(0.022,0.40); ...
```

Example Data

dti

```
## # A tibble: 382 x 5
```

```
##       id sex   case      cca      rcst
##   <dbl> <fct> <fct>      <tfd>      <tfd>
## 1  1001 female contr~ (0.000,0.49);(0.011,0.52~ (0.0000,0.257);(0.0185,0~
## 2  1002 female contr~ (0.000,0.47);(0.011,0.49~ ( 0.222,0.443);( 0.241,0~
## 3  1003 male   contr~ (0.000,0.50);(0.011,0.51~ ( 0.222,0.424);( 0.241,0~
## 4  1004 male   contr~ (0.000,0.40);(0.011,0.42~ (0.0000,0.508);(0.0185,0~
## 5  1005 male   contr~ (0.000,0.40);(0.011,0.41~ ( 0.222,0.398);( 0.241,0~
## 6  1006 male   contr~ (0.000,0.45);(0.011,0.45~ (0.0556,0.467);(0.0741,0~
## 7  1007 male   contr~ (0.000,0.55);(0.011,0.56~ (0.0000,0.519);(0.0185,0~
## 8  1008 male   contr~ (0.000,0.45);(0.011,0.48~ (0.0000,0.333);(0.0185,0~
## 9  1009 male   contr~ (0.000,0.50);(0.011,0.51~ (0.0000,0.568);(0.0185,0~
## 10 1010 male   contr~ (0.000,0.46);(0.011,0.47~ ( 0.222,0.439);( 0.241,0~
## # ... with 372 more rows
```


tf subclass: tfd

tfd objects contain “raw” functional data:

- ▶ represented as a list of **evaluations** $f_i(t)|_{t=t'}$ and corresponding argument vector(s) t'
- ▶ has a **domain**: the range of valid **args**.

```
ex %>% evaluations() %>% str
```

```
## List of 5
```

```
## $ : num [1:93] 0.491 0.516 0.535 0.555 0.594 ...
```

```
## $ : num [1:93] 0.472 0.487 0.502 0.523 0.554 ...
```

```
## $ : num [1:93] 0.502 0.514 0.539 0.573 0.604 ...
```

```
## $ : num [1:93] 0.402 0.422 0.439 0.459 0.476 ...
```

```
## $ : num [1:93] 0.402 0.406 0.399 0.386 0.41 ...
```

```
ex %>% arg() %>% str
```

```
## num [1:93] 0 0.011 0.022 0.033 0.043 0.054 0.065 0.076 0.087 0.098 ...
```

```
ex %>% domain()
```

```
## [1] 0 1
```

tf subclass: tfd

- ▶ each `tfd`-vector contains an `evaluator` function that defines how to inter-/extrapolate evaluations between args (and remembers results of previous calls)

```
evaluator(ex) %>% str

## function (x, arg, evaluations)
##   - attr(*, "memoised")= logi TRUE
##   - attr(*, "class")= chr [1:2] "memoised" "function"

evaluator(ex) = approx_spline
```

tf subclass: tfd

- ▶ tfd has subclasses for regular data with a common grid and irregular or sparse data.

```
dti$rcst[1:2]
```

```
## tfd[2] on (0,1) based on 43 to 55 (mean: 49) evaluations each
```

```
## inter-/extrapolation by approx_linear
```

```
## 1001_1: (0.000,0.26);(0.018,0.45);(0.037,0.40); ...
```

```
## 1002_1: ( 0.22,0.44);( 0.24,0.48);( 0.26,0.48); ...
```

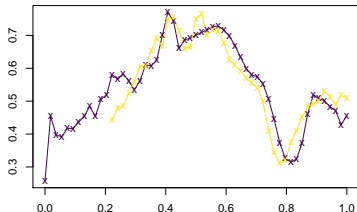
```
dti$rcst[1:2] %>% arg() %>% str
```

```
## List of 2
```

```
## $ 1001_1: num [1:55] 0 0.0185 0.037 0.0556 0.0741 0.0926 0.111 0.13 0.148 0.167
```

```
## $ 1002_1: num [1:43] 0.222 0.241 0.259 0.278 0.296 0.315 0.333 0.352 0.37 0.389
```

```
dti$rcst[1:2] %>% plot(pch = "x", col = viridis(2))
```



tf subclass: tfb

Functional data in basis representation:

- ▶ represented as a list of **coefficients** and a common **basis_matrix** of basis function evaluations on a vector of **arg-values**.
- ▶ contains a **basis** function that defines how to compute the basis for new **args** and how to differentiate/integrate.
- ▶ (internal) flavors: **mgcv**-spline bases and **FPCs** (wavelets to be added)
- ▶ significant memory and time savings:

```
refund::DTI$cca %>% object.size() %>% print(units = "Kb")
```

```
## 304 Kb
```

```
dti$cca %>% object.size() %>% print(units = "Kb")
```

```
## 354.6 Kb
```

```
dti$cca %>% tfb(verbose = FALSE) %>% object.size() %>% print(units = "Kb")
```

```
## 169.9 Kb
```

tf subclass: tfb spline basis

- ▶ accepts all arguments of mgcv's `s()`-syntax
- ▶ either does a penalized fit with (GCV-based) function-specific smoothing or unpenalized.

```
ex_b = ex %>% tfb(); ex_b[1:2]

## Percentage of raw input data variance preserved in basis representation:
## (per functional observation, approx.):

##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##    96.00  96.70   97.60   97.42  98.00   98.80

## tf[2] on (0,1) in basis representation:
## using basis s(arg, bs = "cr", k = 25)
## A: (0.000,0.48);(0.011,0.52);(0.022,0.54); ...
## B: (0.000,0.46);(0.011,0.49);(0.022,0.51); ...

ex[1:2] %>% tfb(bs = "tp", k = 55)

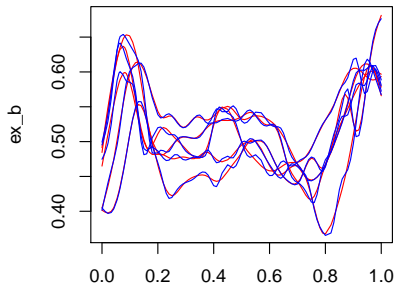
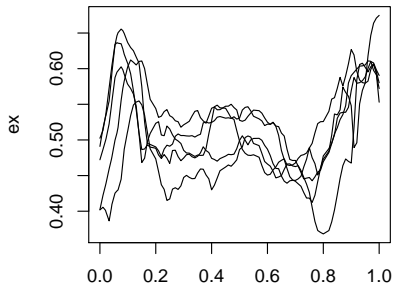
## Percentage of raw input data variance preserved in basis representation:
## (per functional observation, approx.):

##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##    99.50  99.58   99.65   99.65  99.72   99.80

## tf[2] on (0,1) in basis representation:
## using basis s(arg, bs = "tp", k = 55)
## A: (0.000,0.49);(0.011,0.51);(0.022,0.53); ...
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...
```

tf subclass: tfb spline basis

```
ex %>% plot()  
ex_b %>% plot(col = "red")  
ex %>% tfb(k = 35, penalized = FALSE) %>% lines(col = "blue")
```



tfb subclass: tfb FPC-based

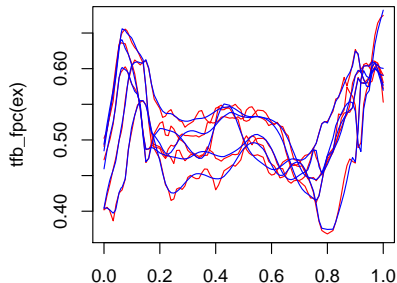
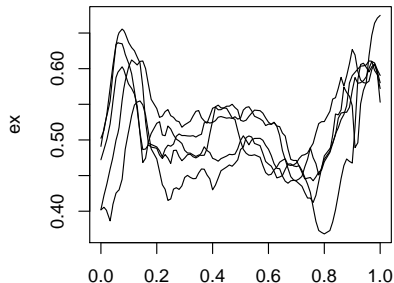
- ▶ uses either
 - ▶ simple unregularized SVD of the data matrix ("smooth = FALSE")
 - ▶ or smoothed covariance estimate from `refund::fpca.sc`
- ▶ corresponding FPC basis and mean function saved as `tfd`-object
- ▶ observed functions are linear combinations of those.

```
(ex %>% tfb_fpc(smooth = FALSE, pve = .999))  
## tfb[5] on (0,1) in basis representation:  
## using basis FPC: 4 components.  
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...  
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...  
## C: (0.000,0.50);(0.011,0.51);(0.022,0.54); ...  
## D: (0.000,0.40);(0.011,0.42);(0.022,0.44); ...  
## E: (0.000,0.40);(0.011,0.41);(0.022,0.40); ...
```

```
(ex %>% tfb_fpc(pve = .95))  
## tfb[5] on (0,1) in basis representation:  
## using basis FPC: 21 components.  
## A: (0.000,0.48);(0.011,0.51);(0.022,0.54); ...  
## B: (0.000,0.46);(0.011,0.49);(0.022,0.51); ...  
## C: (0.000,0.49);(0.011,0.52);(0.022,0.55); ...  
## D: (0.000,0.40);(0.011,0.43);(0.022,0.45); ...  
## E: (0.000, 0.4);(0.011, 0.4);(0.022, 0.4); ...
```

tf subclass: tfb FPC-based

```
ex %>% plot()  
ex %>% tfb_fpc(smooth = FALSE, pve = .999) %>% plot(col = "red")  
ex %>% tfb_fpc(pve = .95) %>% lines(col = "blue")
```



tf-Class: Methods

Subset & subassign

```
ex[1:2]
```

```
## tfd[2] on (0,1) based on 93 evaluations each  
## interpolation by approx_spline  
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...  
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...
```

```
ex[1:2] = ex[2:1]
```

```
ex
```

```
## tfd[5] on (0,1) based on 93 evaluations each  
## interpolation by approx_spline  
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...  
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...  
## C: (0.000,0.50);(0.011,0.51);(0.022,0.54); ...  
## D: (0.000,0.40);(0.011,0.42);(0.022,0.44); ...  
## E: (0.000,0.40);(0.011,0.41);(0.022,0.40); ...
```

Evaluate

```
ex[1:2, seq(0, 1, l = 3)]
```

```
##           0           0.5           1
## B 0.4721627 0.4984125 0.5802742
## A 0.4909345 0.5307563 0.5904773
## attr("arg")
## [1] 0.0 0.5 1.0
```

```
ex["B", seq(0, .15, l = 3), interpolate = FALSE]
```

```
##           0 0.075           0.15
## B 0.4721627    NA 0.4682867
## attr("arg")
## [1] 0.000 0.075 0.150
```

```
ex[1:2, seq(0, 1, l = 2), matrix = FALSE] %>% str
```

```
## List of 2
## $ B:Classes 'tbl_df', 'tbl' and 'data.frame':  2 obs. of  2 variables:
## ..$ arg : num [1:2] 0 1
## ..$ value: num [1:2] 0.472 0.58
## $ A:Classes 'tbl_df', 'tbl' and 'data.frame':  2 obs. of  2 variables:
## ..$ arg : num [1:2] 0 1
## ..$ value: num [1:2] 0.491 0.59
```

Compare & compute

```
ex[1] + ex[1] == 2 * ex[1]
```

```
## [1] TRUE
```

```
log(exp(ex[2])) == ex[2]
```

```
## [1] TRUE
```

```
ex - (2:-2) != ex
```

```
## [1] TRUE TRUE FALSE TRUE TRUE
```

Summarize

```
c(mean = mean(ex), sd = sd(ex))

## tfd[2] on (0,1) based on 93 evaluations each
## interpolation by approx_spline
## mean: (0.000, 0.45);(0.011, 0.47);(0.022, 0.48); ...
## sd: (0.000,0.049);(0.011,0.052);(0.022,0.062); ...

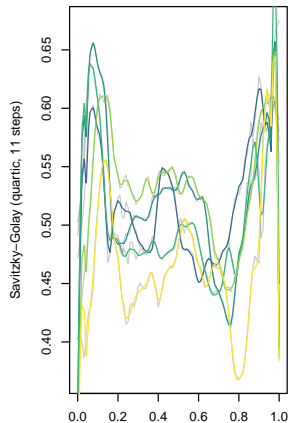
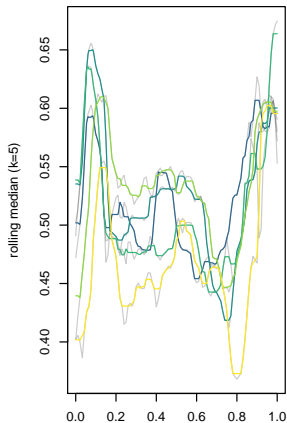
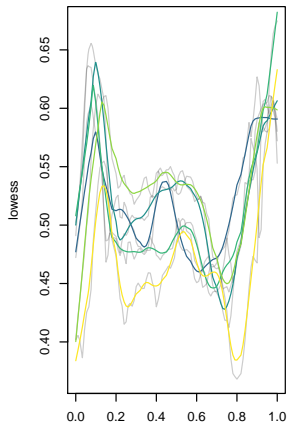
depth(ex) ## Modified Band-2 Depth (à la Sun/Genton/Nychka, 2012), others to come.

##      B      A      C      D      E
## 0.60875 0.66005 0.65805 0.55915 0.51400

median(ex) == ex[which.max(depth(ex))]

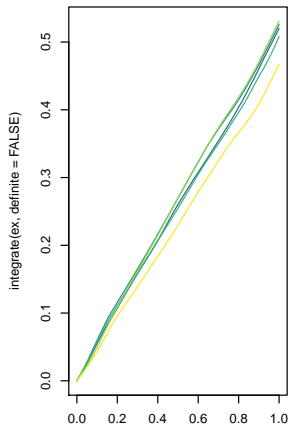
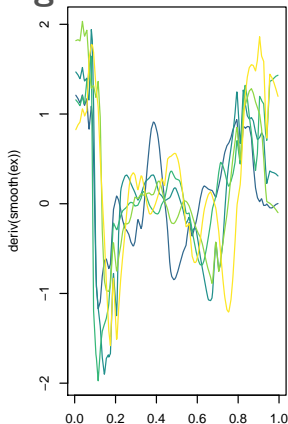
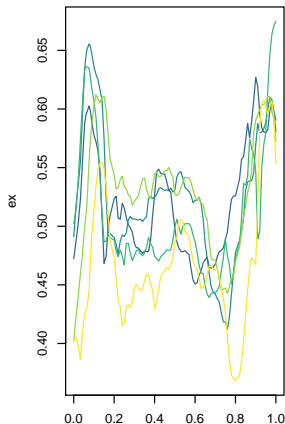
##      A
## TRUE
```

(Simple, local) smoothing



```
ex %>% smooth("lowess") %>% plot
ex %>% smooth("rollmedian", k = 5) %>% plot
ex %>% smooth("savgol", fl = 11) %>% plot
```

Differentiate & integrate



```
ex %>% plot
ex %>% smooth() %>% deriv() %>% plot
ex %>% integrate(definite = FALSE) %>% plot
```

```
ex %>% integrate()
```

```
##           B           A           C           D           E
## 0.5201949 0.5260109 0.5082054 0.5305535 0.4668760
```

Query

Find arguments t satisfying a condition on value $f(t)$ (and argument t):

```
ex %>% anywhere(value > .65)
```

```
##      B      A      C      D      E
## FALSE TRUE  TRUE FALSE FALSE
```

```
ex[1:2] %>% where(value > .6, "all")
```

```
## $B
## [1] 0.076 0.890 0.900 0.910 0.980
##
## $A
## [1] 0.054 0.065 0.076 0.087 0.098 0.110 0.120 0.130 0.140 0.960 0.970
## [12] 0.980
```

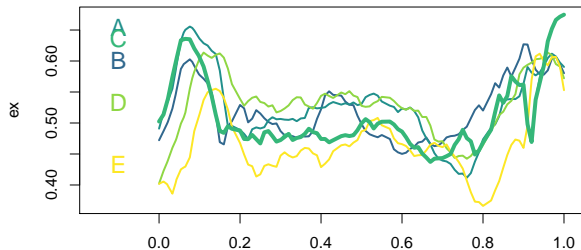
```
ex[2] %>% where(value > .6, "range")
```

```
##   begin  end
## A 0.054 0.98
```

```
ex %>% where(value > .6 & arg > .5, "first")
```

```
##      B      A      C      D      E
## 0.89 0.96 0.96 0.93 0.93
```


Zoom & query



```
ex %>% where(value == max(value), "first")
```

```
##      B      A      C      D      E
## 0.900 0.076 1.000 0.110 0.980
```

```
ex[c("A", "D")] %>% zoom(.5, 1) %>% where(value == max(value), "first")
```

```
##      A      D
## 0.97 0.96
```

```
ex %>% zoom(0.2, 0.6) %>% anywhere(value <= median(ex)[, arg])
```

```
##      B      A      C      D      E
## TRUE TRUE TRUE TRUE TRUE
```

Convert & construct

To & from list, matrix or data frame with "id","arg","value"-columns:

```
ex_matrix = ex %>% as.matrix(); ex_matrix[1:2, 1:3]

##           0      0.011      0.022
## B 0.4721627 0.4866481 0.5018916
## A 0.4909345 0.5164951 0.5352068

ex_df = ex %>% as.data.frame(); str(ex_df)

## Classes 'tbl_df', 'tbl' and 'data.frame':   465 obs. of  3 variables:
## $ id   : Ord.factor w/ 5 levels "B"<"A"<"C"<"D"<...: 1 1 1 1 1 1 1 1 1 1 ...
## $ arg  : num  0 0.011 0.022 0.033 0.043 0.054 0.065 0.076 0.087 0.098 ...
## $ value: num  0.472 0.487 0.502 0.523 0.554 ...

ex_matrix[1:2, ] %>% tfd()

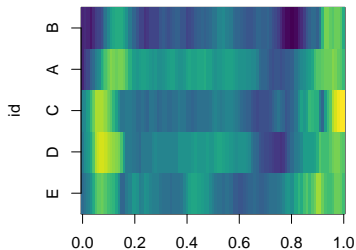
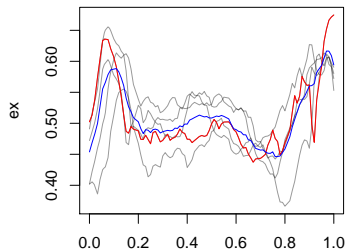
## tfd[2] on (0,1) based on 93 evaluations each
## interpolation by approx_linear
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...

tfd(ex_df) == tfd(ex_matrix)

##      B      A      C      D      E
## TRUE TRUE TRUE TRUE TRUE
```

Visualize: base

```
layout(t(1:2))  
plot(ex, type = "spaghetti"); lines(c(median(ex), mean(ex)), col = c(2, 4))  
plot(ex, type = "lasagna", col = viridis(50))
```



Visualize: ggplot2

New **pasta-themed** geoms with a **tf**-aesthetic for functional data:

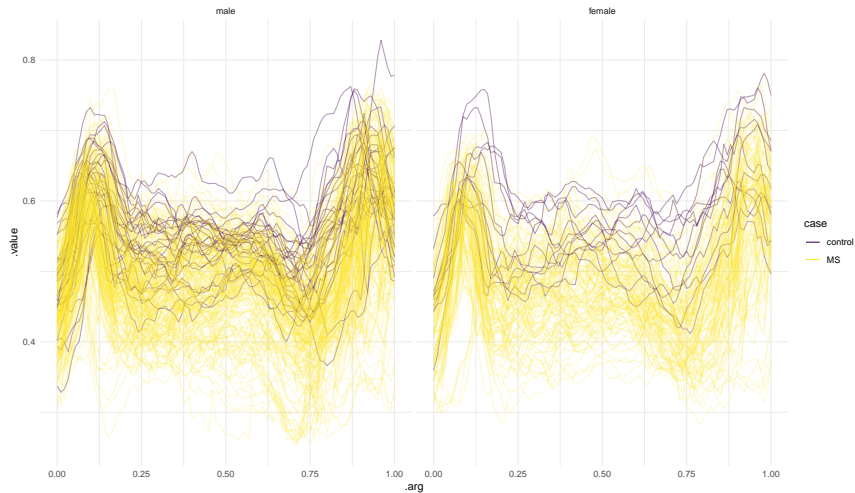
- ▶ `geom_spaghetti` for lines
- ▶ `geom_meatballs` for (lines &) points
- ▶ `geom_lasagna` with an `order`-aesthetic to sort the lasagna layers

To come:

- ▶ `geom_pappardelle` for functional boxplots
- ▶ `geom_capellini` for little sparklines / glyphs on maps etc.

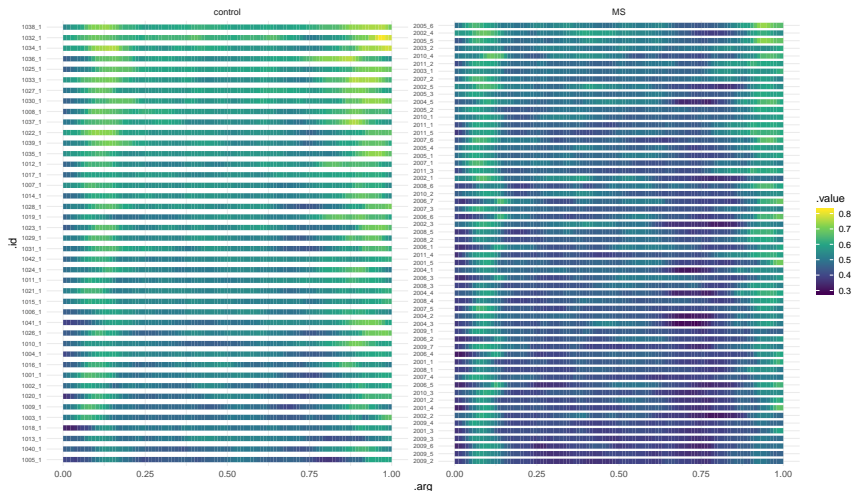
Visualize: ggplot2

```
ggplot(dti, aes(tf = cca, colour = case)) +  
  geom_spaghetti() + facet_wrap(~ sex)
```



Visualize: ggplot2

```
ggplot(dti, aes(tf = cca, order = integrate(cca, definite = TRUE))) +  
  geom_lasagna() + facet_wrap(~ case)
```



Wrangling tfs inside data frames

Wrangling tfs inside data frames: dplyr

dplyr verbs `filter`, `select`, `mutate`, `summarize` work on tf-columns - e.g.:

group-wise functional means:

```
dti %>% group_by(case, sex) %>% summarize(mean_rcst = mean(rcst, na.rm = TRUE))
```

```
## # A tibble: 4 x 3
```

```
##   case    sex                mean_rcst
##   <fct>  <fct>                <tfd>
## 1 control male (0.0000,0.514);(0.0185,0.50...
## 2 control female (0.0000,0.517);(0.0185,0.53...
## 3 MS      male (0.0000,0.533);(0.0185,0.52...
## 4 MS      female (0.0000,0.524);(0.0185,0.51...
```

which subjects go below cca = .26:

```
dti %>% filter(anywhere(cca, value < .26))
```

```
## # A tibble: 3 x 5
```

```
##   id sex case                cca                rcst
##   <dbl> <fct> <fct>                <tfd>                <tfd>
## 1  2017 male MS (0.000,0.38);(0.011,0.38);~ (0.0741,0.519);(0.0926,0.~
## 2  2017 male MS (0.000,0.34);(0.011,0.35);~ (0.0000,0.616);(0.0185,0.~
## 3  2083 male MS (0.000,0.39);(0.011,0.43);~ (0.0000,0.511);(0.0185,0.~
```


Wrangling tfs inside data frames: dplyr

```
# center & scale functional data:
```

```
dti %>%
```

```
  mutate(cca = tfb(cca, verbose = FALSE),  
         cca_z = (cca - mean(cca))/sd(cca)) %>% glimpse
```

```
## Observations: 382
```

```
## Variables: 6
```

```
## $ id      <dbl> 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 10...
```

```
## $ sex     <fct> female, female, male, male, male, male, male, male, male...
```

```
## $ case    <fct> control, control, control, control, control, control, co...
```

```
## $ cca     <tf> 1001_1: (0.000,0.48);(0.011,0.52);(0.022,0.54); ..., 1002...
```

```
## $ rcst    <tfd> 1001_1: (0.000,0.26);(0.018,0.45);(0.037,0.40); ..., 100...
```

```
## $ cca_z   <tf> [1]: (0.000, 0.84);(0.011, 0.98);(0.022, 1.10); ..., [...
```

Wrangling tfs inside data frames: tidyr

tidyfun provides `tf_` variants of `tidyr`-verbs to reshape and reformat functional data while keeping it in sync with other covariates:

- ▶ `tf_spread`: `tf` \rightarrow columns for each arg
- ▶ `tf_gather`: columns for each arg \rightarrow `tf`
- ▶ `tf_nest` : data in long format (`id`, `arg`, `value`) \rightarrow `tf`
- ▶ `tf_unnest`: `tf` \rightarrow data in long format (`id`, `arg`, `value`)

Wrangling tfs inside data frames: tidyr

spread tf out into columns for each arg

```
dti_wide = dti %>% tf_spread(cca); dti_wide[, 1:7] %>% glimpse()
```

```
## Observations: 382
```

```
## Variables: 7
```

```
## $ id      <dbl> 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009...
## $ sex     <fct> female, female, male, male, male, male, male, male, ...
## $ case    <fct> control, control, control, control, control, control, control...
## $ rcst     <tfd> 1001_1: (0.000,0.26);(0.018,0.45);(0.037,0.40); ...,...
## $ cca_0    <dbl> 0.4909345, 0.4721627, 0.5023738, 0.4021894, 0.401874...
## $ cca_0.011 <dbl> 0.5164951, 0.4866481, 0.5135178, 0.4222717, 0.405536...
## $ cca_0.022 <dbl> 0.5352068, 0.5018916, 0.5386470, 0.4394859, 0.398715...
```

collect all columns into a single tf-column

(... will try to guess arg from column names, name of tf from their prefix)

```
dti_wide %>% tf_gather(matches("cca_")) %>% glimpse()
```

```
## Observations: 382
```

```
## Variables: 5
```

```
## $ id      <dbl> 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 101...
## $ sex     <fct> female, female, male, male, male, male, male, male, male,...
## $ case    <fct> control, control, control, control, control, control, control, con...
## $ rcst     <tfd> 1001_1: (0.000,0.26);(0.018,0.45);(0.037,0.40); ..., 1002...
## $ cca      <tfd> [1]: (0.000,0.49);(0.011,0.52);(0.022,0.54); ..., [2]: (0...
```

Wrangling tfs inside data frames: tidyr

```
# unnest tf by writing 3 loong columns id, arg, value:
# (will try to avoid unnecessary duplication of columns)
dti_long = dti %>% tf_unnest(cca); dti_long %>% glimpse()

## Observations: 35,526
## Variables: 7
## $ id          <dbl> 1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001...
## $ sex         <fct> female, female, female, female, female, female, fema...
## $ case        <fct> control, control, control, control, control, control...
## $ rcst        <tfd> 1001_1: (0.000,0.26);(0.018,0.45);(0.037,0.40); ...,...
## $ cca_id       <chr> "1001_1", "1001_1", "1001_1", "1001_1", "1001_1", "1...
## $ cca_arg      <dbl> 0.000, 0.011, 0.022, 0.033, 0.043, 0.054, 0.065, 0.0...
## $ cca_value    <dbl> 0.4909345, 0.5164951, 0.5352068, 0.5546577, 0.594497...
```

```
# nest tf by writing 3 loong columns id, arg, value:
dti_long %>% tf_nest(cca_value, .id = cca_id, .arg = cca_arg) %>% glimpse()

## Observations: 382
## Variables: 6
## $ cca_id       <chr> "1001_1", "1002_1", "1003_1", "1004_1", "1005_1", "1...
## $ id          <dbl> 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009...
## $ sex         <fct> female, female, male, male, male, male, male, ...
## $ case        <fct> control, control, control, control, control, control...
## $ rcst        <tfd> [1]: (0.000,0.26);(0.018,0.45);(0.037,0.40); ..., [2...
## $ cca_value    <tfd> 1001_1: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...,...
```

Fin

Summary

... I like it. You might, too, give it a spin!¹

`https://github.com/fabian-s/tidyfun`

Get it: `devtools::install_github("fabian-s/tidyfun")`

¹*Currently a moving target. Still early β , some kinks need to be worked out.*

Outlook

Next up:

- ▶ integrate fda bases & penalties, wavelet bases
- ▶ improve & extend ggplot2 interface
- ▶ functions for registering & warping
- ▶ validate & improve

Version 1.0:

- ▶ extensions for multivariate functions
- ▶ much more extensive documentation & tests
- ▶ integration with **refund** for modeling and inference

Thanks.

`https://github.com/fabian-s/tidyfun`

I don't even have references.