`tidyfun`: Tidy Functional Data

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

Fabian Scheipl[1] Arthur Jeff Goldsmith[2]

[1]: Dept. of Statistics, LMU Munich

[2]: Columbia University Mailman School of Public Health

JSM 2018

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Emmy Noether-Programm
Deutsche Forschungsgemeinschaft
DFG

tidyfun

The goal of tidyfun is to provide a tidyverse-compliant, accessible and well-documented way to deal with functional data in R, specifically for data wrangling and exploratory analysis.

tidyfun provides:

- new R 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.

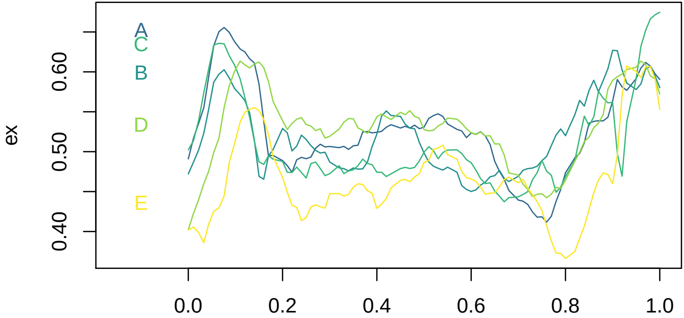
`tf`-Class: Definition

`tf`-class

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

- abstract superclass for functional data
  - as (argument, value)-tuples: subclass `tfd`, 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 help define *function-like* behavior:
  - how to **evaluate** the given ‘functions’ for new arguments
  - their **domain**
  - the **resolution** of the argument values

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:

- a list of **`evaluations`** $f_i(t)|_{t=t'}$ and corresponding **`args`** t'
- the **`domain`**: the range of valid **`args`**.

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

```
## List of 5
##  $ : num [1:93] 0.491 0.517 0.536 0.555 0.593 ...
##  $ : num [1:93] 0.472 0.487 0.502 0.523 0.552 ...
##  $ : num [1:93] 0.502 0.514 0.539 0.574 0.603 ...
##  $ : num [1:93] 0.402 0.423 0.44 0.46 0.475 ...
##  $ : num [1:93] 0.402 0.406 0.399 0.386 0.409 ...
```

```
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

- a modifiable **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

- internal subclasses for regular tfd with a common grid and irregular tfd.

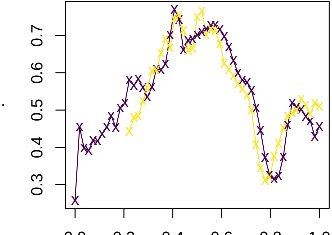
```
dti$rcst[1:2]
```

```
## 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:

- keep a list of **`coefficients`** and a corresponding common **`basis_matrix`** of basis function evaluations
- have an associated **`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 savings for large data:

```
dti$cca %>% object.size()
```

```
## 783456 bytes
```

```
dti$cca %>% tfb(verbose = FALSE) %>% object.size()
```

```
## 174000 bytes
```

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.
##   95.50   96.40   96.80   97.04   97.80   98.70

## tf[2] on (0,1) in basis representation:
##  using basis  s(arg, bs = "cr", k = 25)
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...
## B: (0.000,0.47);(0.011,0.49);(0.022,0.51); ...
```

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

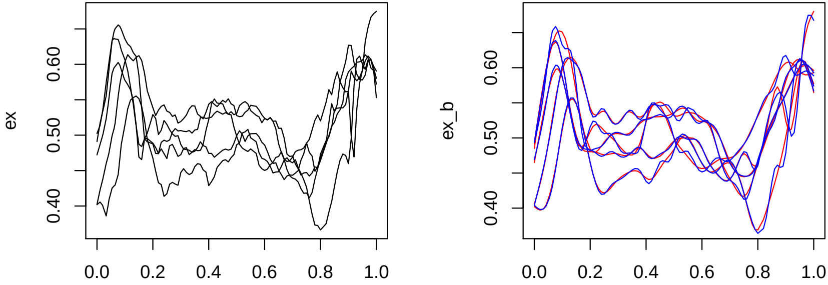
```
## Percentage of raw input data variance preserved in basis representation:
## (per functional observation, approx.):

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    99.2    99.3    99.4    99.4    99.5    99.6

## 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);(0.022,0.54); ...
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50)
```

tf subclass: tfb spline basis

```
plot(ex, alpha = 1)
plot(ex_b, col = "red")
lines(ex %>% tfb(penalized = FALSE, k = 30), col = "blue")
```



`tf` 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: 19 components.
## A: (0.000,0.49);(0.011,0.51);(0.022,0.54); ...
## B: (0.000,0.46);(0.011,0.49);(0.022,0.51); ...
## C: (0.000,0.50);(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-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 0.4690637
## 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); ...
```

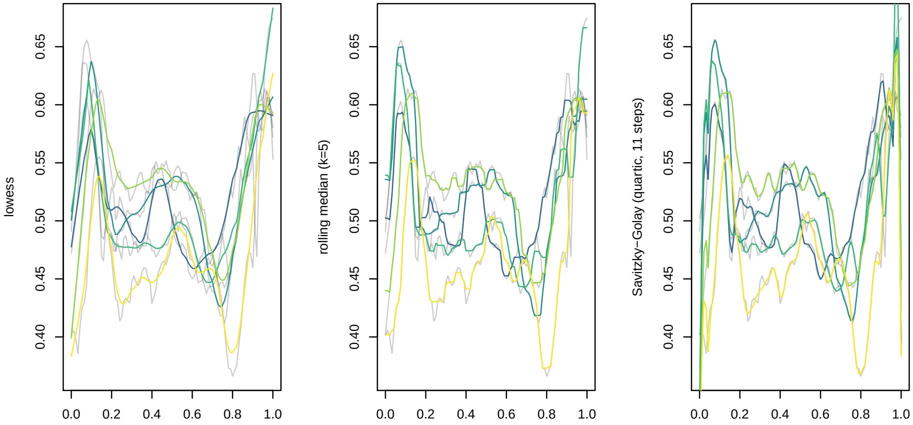
```
depth(ex) ## Modified Band-2 Depth
```

```
##       B       A       C       D       E
## 0.61125 0.64955 0.66055 0.56815 0.51050
```

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

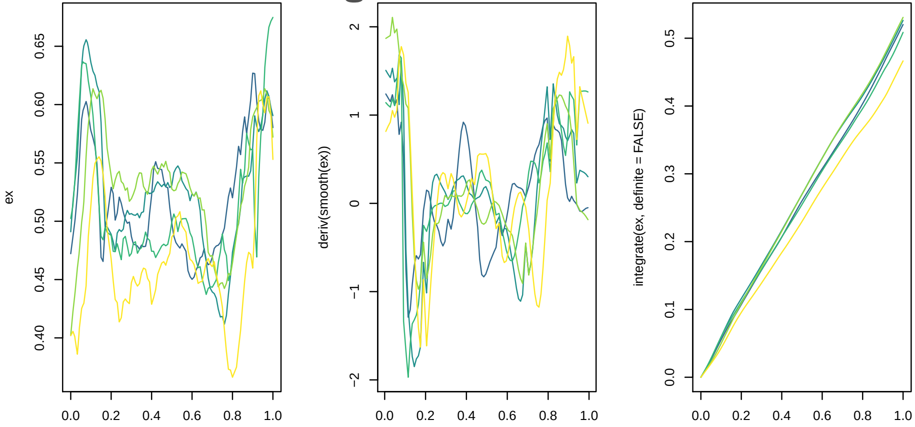
```
## [1] FALSE
```

(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
## 0.5202133 0.5263170 0.5085679 0.5307260 0.4665386
```

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 0.920 0.970 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
```

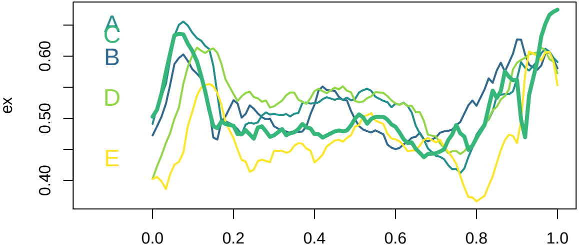
```
ex[2] %>% where(value > .6, "range")
```

```
##   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 0.930
```

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

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

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

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

Convert & construct

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

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

```
##           0     0.011     0.022
## B 0.4721627 0.4868219 0.5022577
## A 0.4909345 0.5168018 0.5356539
```

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

```
## 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 0.552 ...
```

```
m_ex[1:2, ] %>% tfd()
```

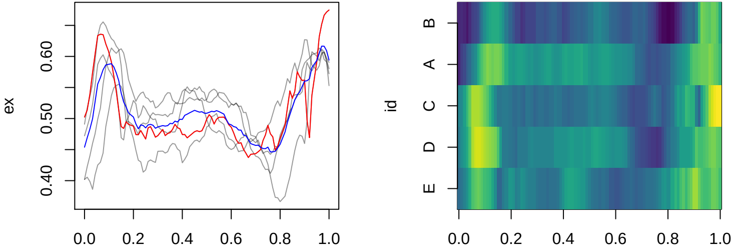
```
## tfd[2] on (0,1) based on 93 evaluations each
## interpolation by approx_linear
## B: (0.000,0.47);(0.011,0.49);(0.022,0.50); ...
## A: (0.000,0.49);(0.011,0.52);(0.022,0.54); ...
```

```
tfd(df_ex) == tfd(m_ex)
```

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

Visualize: `base`

```r
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))
```



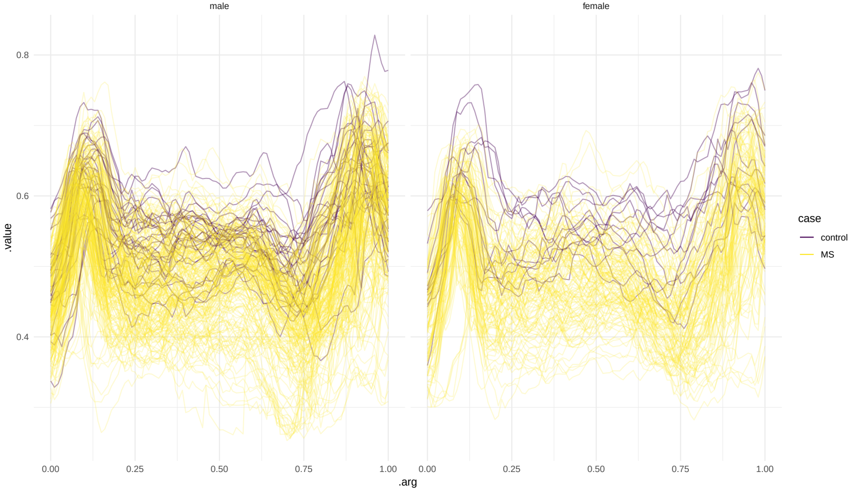
Visualize: `ggplot2`

New 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

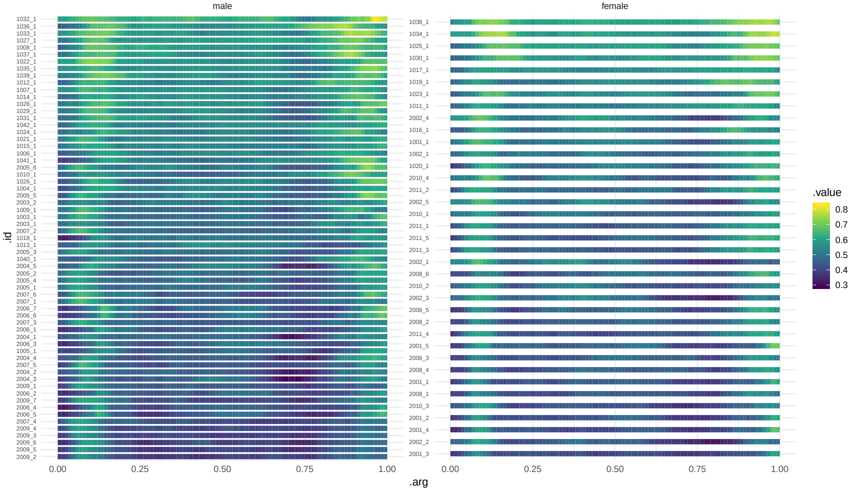
Visualize: ggplot2

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



Visualize: ggplot2

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



TODOs:

```r
head(dti)
dim(dti)
nrow(dti %>%  filter(rcst[, .7] > .8))

plot(dti$cca, points = FALSE)
lines(mean(dti$cca), col = "red")
lines(mean(dti$cca) + sd(dti$cca), col = "blue", lty = 2)
lines(mean(dti$cca) - sd(dti$cca), col = "blue", lty = 2)

plot(dti$rcst, type = "lasagna")
funplot(dti$rcst, type = "lasagna")
funplot(dti$rcst) + theme_minimal()

## to come:
## dti %>% group_by(sex) %>% mutate(mean_cca = mean(cca), sd_cca = sd(cca))
```

- derivatives: might be fairly easy for `tfb` since `mgcv` offers derivatives of its bases
- registering/warping should be mostly easy, just overwrite `argvals` (or wrap warping around `evaluator`...?)
- intensive exing with diverse use-cases
- extensions for multivariate and image data (will be hard)
- integration with renovated `refund` for modeling etc.

ISSUES:

- lots of `tibblyverse` adjustments still needed (no grouped operations possible ATM, no pretty printing)
- is `signif_argvals` reasonable?
- no `S4` means no multiple inheritance for orthogonal implementation of aspects "representation" and "function properties" like monotonous or strictly positive functions in basis or raw data representation.
- more issues: [https://github.com/fabian-s/tidyfun/issues] –>