Timing Anticipation Analysis Pipeline

D. Carioti

Overview

This is the analysis pipeline for the Anticipation task included in the paper “Anticipating the beat: Rhythm and Meter in Developmental Dyslexia” by Carioti et al.

The pipeline aims at clearly separate preprocessing, quality control, statistical testing

Anticipation Dataset Uploading

Participant-level Descriptive Statistics

	item	Condition	Part.	vars	n	mean	sd	median	trimmed
X11	1	test	ddARS002	1	10	267.5000	274.49762	167.5	222.6250
X12	2	training	ddARS002	1	5	569.4000	158.41496	595.0	569.4000
X13	3	test	ddARS003	1	10	58.5000	160.00990	19.5	38.3750
X14	4	training	ddARS003	1	6	840.3333	16448.58859	-4494.5	840.3333
X15	5	test	ddARS004	1	11	1043.7273	25456.43932	-5325.0	-6094.2222
X16	6	training	ddARS004	1	7	3469.4286	18272.04056	-5344.0	3469.4286
X17	7	test	ddARS005	1	12	9060.8333	30028.45420	-524.5	4807.2000
X18	8	training	ddARS005	1	15	5518.6667	17980.16024	-2007.0	4863.3846
X19	9	test	ddARS006	1	12	5212.4167	33312.17629	-6142.0	360.1000
X110	10	training	ddARS006	1	7	7611.7143	17807.16353	-46.0	7611.7143
X111	11	test	ddARS007	1	9	7545.0000	2983.88673	9413.0	7545.0000
X112	12	training	ddARS007	1	8	7164.2500	19639.27372	-3282.0	7164.2500
X113	13	test	ddARS008	1	10	202.8000	196.00612	160.0	167.3750
X114	14	training	ddARS008	1	15	6737.4000	17464.15083	257.0	6269.0000
X115	15	test	ddARS009	1	9	2505.5556	4137.52846	31.0	2505.5556
X116	16	training	ddARS009	1	12	977.1667	14606.49217	35.0	-797.9000
X117	17	test	ddARS010	1	12	10879.5000	30305.22749	92.5	6556.3000
X118	18	training	ddARS010	1	11	514.6364	11448.49419	67.0	-2032.6667
X119	19	test	ddARS011	1	11	850.1818	25424.51881	-5523.0	-6290.6667
X120	20	training	ddARS011	1	7	2282.7143	19174.72288	-5006.0	2282.7143
X121	21	test	ddARS012	1	11	987.5455	25392.78956	-5596.0	-6105.5556
X122	22	training	ddARS012	1	8	3523.8750	20371.99295	-3403.5	3523.8750
X123	23	test	ddARS013	1	10	142.5000	86.60928	126.5	141.0000
X124	24	training	ddARS013	1	5	484.0000	472.48862	339.0	484.0000
X125	25	test	ddARS014	1	10	103.2000	575.45341	136.0	149.3750
X126	26	training	ddARS014	1	6	3443.3333	14551.34443	-511.5	3443.3333
X127	27	test	ddARS015	1	12	10154.0000	30481.64830	21.0	5438.8000
X128	28	training	ddARS015	1	7	2564.7143	19101.52580	-4898.0	2564.7143
X129	29	test	ddARS016	1	12	5971.9167	32482.22924	-7254.5	682.3000
X130	30	training	ddARS016	1	7	4288.1429	17703.09537	-3327.0	4288.1429
X131	31	test	ddARS017	1	10	117.4000	92.86573	104.5	112.7500
X132	32	training	ddARS017	1	7	2318.0000	19147.78732	-5366.0	2318.0000
X133	33	test	ddARS018	1	12	7571.9167	31547.80358	-4260.5	2646.4000
X134	34	training	ddARS018	1	8	11000.1250	18965.61218	368.0	11000.1250

```
X135   35     test ddARS019    1 10    46.1000    93.11218    71.5    70.3750
X136   36 training ddARS019    1  8  2833.3750 20786.45661 -2983.5  2833.3750
X137   37     test ddARS020    1 12  9626.4167 31033.76742   222.0  5140.7000
X138   38 training ddARS020    1  9  9455.2222 22078.44007 -1475.0  9455.2222
X139   39     test ddARS021    1 10   240.0000   262.80157   322.0   284.8750
X140   40 training ddARS021    1 16  8117.1875 17483.91990    29.0  7767.5000
X141   41     test ddARS022    1 11  1734.9091 25268.77704 -5403.0 -5226.3333
X142   42 training ddARS022    1  9  6028.5556 18788.58511 -1521.0  6028.5556
X143   43     test ddARS023    1 10    42.4000    42.38501    38.0    38.8750
X144   44 training ddARS023    1  7  2300.4286 19011.63478 -5635.0  2300.4286
X145   45     test ddARS024    1 11  1742.7273 25208.41578 -5274.0 -5208.2222
X146   46 training ddARS024    1  9  6464.3333 19597.11764 -1498.0  6464.3333
X147   47     test ddARS025    1 10    54.0000    63.25082    47.5    43.8750
X148   48 training ddARS025    1 13  1914.0769 16244.63736    86.0  1220.7273
X149   49     test ddARS026    1 12  5949.5833 32493.70724 -7290.0   739.5000
X150   50 training ddARS026    1  9  5959.2222 18723.95560 -1543.0  5959.2222
X151   51     test ddARS027    1 11  4628.1818 24235.85148   -31.0 -1839.6667
X152   52 training ddARS027    1 12  4003.3333 15470.92886    16.5  2734.3000
X153   53     test ddARS028    1 11   789.4545 25433.06529 -5569.0 -6331.6667
X154   54 training ddARS028    1  6   484.8333 16516.04492 -4565.0   484.8333
X155   55     test ddARS029    1 10    76.2000    61.08428    70.0    76.6250
X156   56 training ddARS029    1 23  9431.9565 19681.02537 11530.0 10081.4211
X157   57     test ddARS030    1  9  7701.1111  2694.25706  9389.0  7701.1111
X158   58 training ddARS030    1 13  5798.5385 16806.49058    -6.0  5116.9091
X159   59     test ddARS031    1 11  3457.3636 24790.53960 -2960.0 -3165.5556
X160   60 training ddARS031    1  4  7499.2500  2837.39968  7986.0  7499.2500
X161   61     test ddARS032    1 13  7669.7692 36798.94051 -9165.0  3714.7273
X162   62 training ddARS032    1 14  8615.5714 16769.90754  -483.0  7926.0833
X163   63     test ddARS033    1 10    48.3000    35.31147    40.0    48.1250
X164   64 training ddARS033    1 14  5431.7857 17363.36159 -1325.5  4615.7500
X165   65     test ddARS034    1 10   578.1000   140.78624   620.5   599.8750
X166   66 training ddARS034    1 10    21.8000    46.33405    20.0    20.5000
X167   67     test ddARS035    1 12  7116.2500 32380.92184 -1817.5  2703.7000
X168   68 training ddARS035    1  7  1515.5714 19605.90952 -5595.0  1515.5714
X169   69     test ddARS036    1  9  2497.0000  4090.36591    19.0  2497.0000
X170   70 training ddARS036    1 11  1721.5455 11619.02930   -33.0  -608.7778
X171   71     test ddARS037    1 11   866.8182 25408.52811 -5549.0 -6248.2222
X172   72 training ddARS037    1 10  3020.8000 11765.58002   256.5   722.2500
X173   73     test ddARS038    1 10    69.1000   161.32331    48.5    45.7500
X174   74 training ddARS038    1 14  5459.5714 17955.47021   193.5  4626.2500
X175   75     test ddARS039    1 10   203.2000   300.69024   262.5   257.5000
X176   76 training ddARS039    1 13  2684.8462 16876.15990 -1480.0  1265.8182
X177   77     test ddARS040    1  9  5023.7778  4320.15155  4128.0  5023.7778
X178   78 training ddARS040    1  6  1858.8333 16102.30911 -1905.0  1858.8333
            mad    min   max range        skew    kurtosis          se
X11    142.3296     22   872   850  1.11650714 -0.24056806    86.80377
X12    220.9074    385   744   359 -0.08873845 -2.13759461    70.84532
X13    106.7472    -85   363   448  0.93749970 -0.78039710    50.59957
X14   5610.8997  -9352 33820 43172  1.25485278 -0.25590841  6715.10817
X15   5421.8682 -10931 77260 88191  2.40433256  4.31883238  7675.40525
X16   5573.0934 -12886 34123 47009  0.71376901 -1.46567093  6906.18218
```

X17	2833.9899	-12878	73536	86414	1.48634001	0.39950733	8668.46806
X18	10363.3740	-14597	34153	48750	0.60605038	-1.47068489	4642.45741
X19	9828.8967	-18308	77256	95564	1.44422904	0.32199050	9616.39697
X110	4507.1040	-8849	34284	43133	0.67590351	-1.61192655	6730.47518
X111	3150.5250	3950	11538	7588	-0.13920187	-1.92570959	994.62891
X112	11249.2275	-12670	34029	46699	0.39735172	-1.91525052	6943.53181
X113	161.6034	5	684	679	1.29691386	0.83554714	61.98258
X114	16479.0990	-14624	34188	48812	0.50011568	-1.43478525	4509.22435
X115	130.4688	-90	9626	9716	0.92358573	-1.09647876	1379.17615
X116	5180.9457	-14437	34142	48579	1.08024429	0.04527741	4216.53109
X117	61.5279	-12058	77049	89107	1.50610580	0.44220765	8748.36562
X118	2188.3176	-9269	33224	42493	1.99484870	3.15688560	3451.85088
X119	5285.4690	-10990	76958	87948	2.40294777	4.31393492	7665.78085
X120	11506.4586	-12917	34003	46920	0.68432876	-1.51488011	7247.36403
X121	4919.2668	-11162	76975	88137	2.40001134	4.30452930	7656.21412
X122	20453.2083	-21443	33931	55374	0.29210328	-1.78250103	7202.58718
X123	100.8168	18	279	261	0.08894846	-1.40995033	27.38826
X124	63.7518	130	1315	1185	0.97992102	-1.01479059	211.30334
X125	119.3493	-1272	1109	2381	-0.82441374	1.19691647	181.97434
X126	2452.2204	-8392	32391	40783	1.20013931	-0.30841809	5940.56149
X127	3642.0069	-9412	76872	86284	1.52397617	0.46402399	8799.29393
X128	11613.2058	-12854	34039	46893	0.66145712	-1.53059366	7219.69813
X129	3969.6615	-12356	77196	89552	1.54776586	0.49888166	9376.81190
X130	8255.1168	-10449	33994	44443	0.72771675	-1.46741254	6691.14111
X131	88.2147	-20	292	312	0.37329142	-1.02828236	29.36672
X132	10118.7450	-12698	34095	46793	0.69653943	-1.50696068	7237.18334
X133	7002.3198	-12768	77167	89935	1.50676625	0.43858761	9107.06644
X134	9567.2178	-8962	34447	43409	0.36185124	-1.99291155	6705.35649
X135	36.3237	-211	109	320	-2.03120532	2.85512372	29.44466
X136	22843.1595	-22205	33787	55992	0.27346944	-1.79247639	7349.12221
X137	2913.3090	-12986	77096	90082	1.48414883	0.40179541	8958.67699
X138	16955.0136	-12911	37618	50529	0.17890648	-2.04095664	7359.48002
X139	94.8864	-396	517	913	-1.34885801	0.67551814	83.10515
X140	13834.8819	-12824	33954	46778	0.42488257	-1.66484579	4370.97998
X141	5746.5576	-11027	77148	88175	2.37627575	4.22928700	7618.82294
X142	19399.8210	-14606	33912	48518	0.19771656	-1.89939156	6262.86170
X143	26.6868	-22	135	157	0.70118973	-0.07221200	13.40332
X144	10857.0798	-12977	33768	46745	0.68734995	-1.51130600	7185.72252
X145	5659.0842	-10929	76973	87902	2.37600323	4.22826157	7600.62334
X146	19159.6398	-14421	33574	47995	0.14990507	-1.99362442	6532.37255
X147	32.6172	-26	215	241	1.40630857	1.47740514	20.00167
X148	8476.0242	-22386	33841	56227	0.56714130	-0.76811580	4505.45177
X149	5274.3495	-13018	77018	90036	1.53535289	0.47618078	9380.12531
X150	19497.6726	-14694	33722	48416	0.19513002	-1.89918636	6241.31853
X151	84.5082	-9394	76861	86255	2.36076620	4.19662818	7307.38417
X152	14381.2200	-13095	33792	46887	0.63745092	-1.09756609	4466.07247
X153	5547.8892	-11240	76909	88149	2.40129998	4.30885062	7668.35771
X154	4129.7823	-9289	33795	43084	1.28808908	-0.19954029	6742.64710
X155	91.1799	-9	158	167	0.01418415	-1.83292294	19.31654
X156	25174.5480	-23953	34285	58238	-0.16297617	-1.57741947	4103.77747
X157	2849.5572	4114	11311	7197	-0.17640360	-1.83654347	898.08569

```
X158 16047.6624 -14662 33757 48419  0.44759418 -1.39070770  4661.28181
X159  5138.6916 -10683 77204 87887  2.34622462  4.14199308  7474.62893
X160  2761.3425   4132  9893  5761 -0.15955993 -2.25655359  1418.69984
X161  8232.8778 -18381 77226 95607  1.10818215 -0.73121098 10206.18976
X162  9561.2874  -9101 34606 43707  0.66787811 -1.41716828  4481.94632
X163    46.7019     -6   104   110  0.07285245 -1.48425013    11.16647
X164  8891.1522 -13157 33813 46970  0.74545677 -1.28065776  4640.55358
X165    31.8759    252   730   478 -1.22335429  0.24313429    44.52052
X166    57.8214    -37    91   128  0.09851721 -1.65374945    14.65211
X167  9693.2388 -18637 76995 95632  1.43721492  0.32313812  9347.56697
X168 11294.4468 -14681 33819 48500  0.66468489 -1.52836882  7410.33726
X169    60.7866    -42  9408  9450  0.90541683 -1.13123405  1363.45530
X170  2309.8908  -9313 33729 43042  1.75269083  2.38540459  3503.26915
X171  5258.7822 -11155 76924 88079  2.40256331  4.31298336  7660.95947
X172  1362.5094  -9336 33766 43102  1.65556321  1.80927615  3720.60309
X173   147.5187    -88   413   501  0.91664790 -0.48868978    51.01491
X174 15113.6244 -13141 34060 47201  0.66809598 -1.31144946  4798.80127
X175   131.2101   -571   543  1114 -1.50208994  1.60407683    95.08660
X176 10993.4790 -13203 34182 47385  0.95150917 -0.76165680  4680.60460
X177  5552.3370    119 11572 11453  0.16943243 -1.71076006  1440.05052
X178  6805.1340  -9445 33686 43131  1.17004649 -0.37703469  6573.74017
```

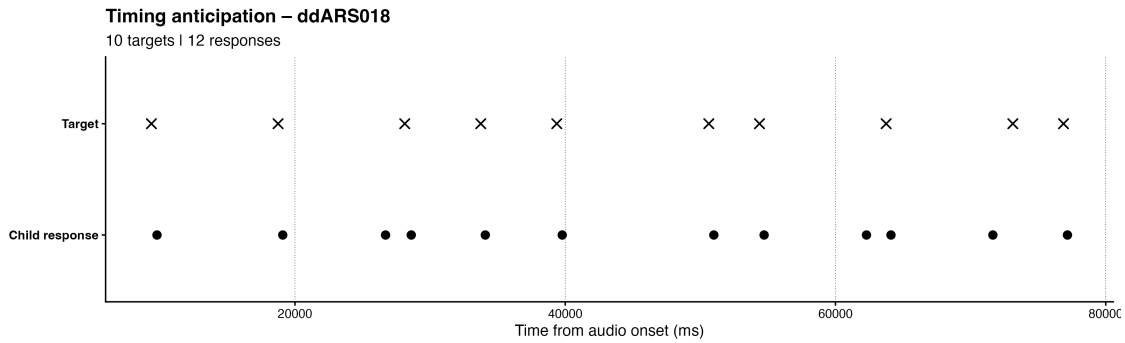
Plots for participant level:

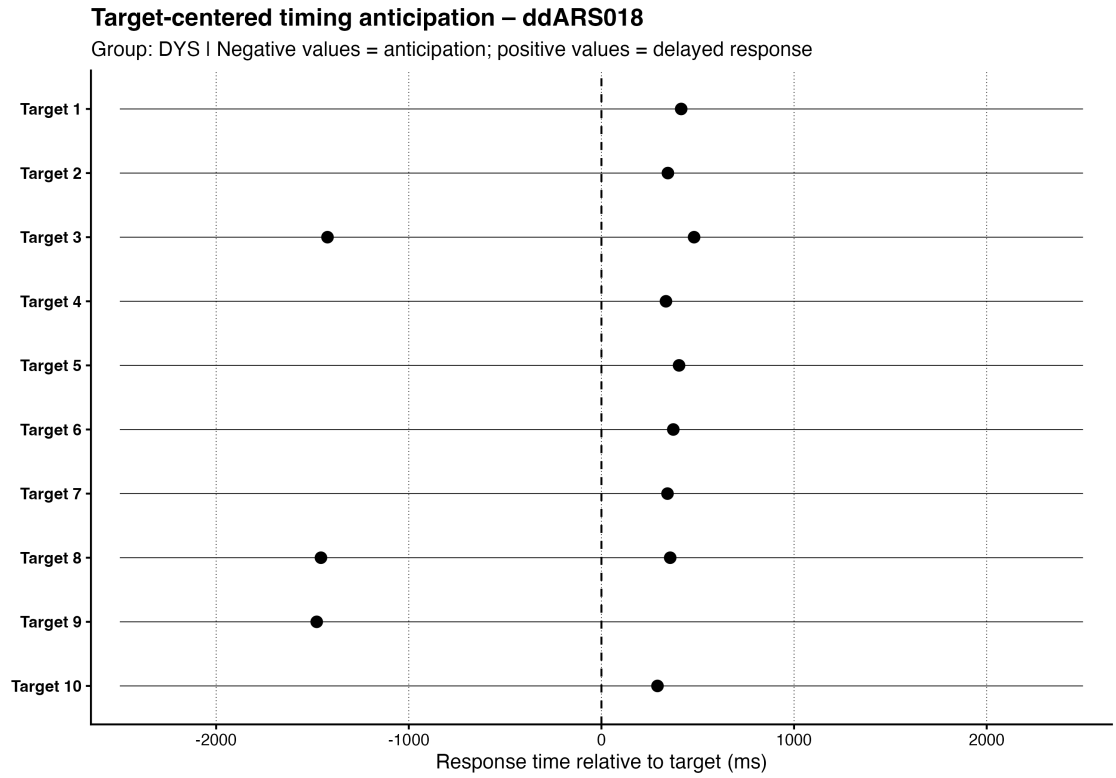
we will save some plots for participants' performance

Should we need a realignment? Let's now consider the performance of a participant:

```
# A tibble: 12 × 4
   out_iresp out_tref out_tresp out_tdelta
 1         1     9375      9789        414
 2         2    18750     19095        345
 3         3    28125     26703      -1422
 4         4    33750     28606      -5144
 5         5    39375     34085      -5290
 6         6    50625     39778     -10847
 7         7    54375     50998      -3377
 8         8    63750     54718      -9032
 9         9    73125     62294     -10831
10        10    76875     64107     -12768
11        11       -1     71647      71648
12        12       -1     77166      77167
```

The fifth and seventh clicks should be considered as referred to the same target event.





Let's now consider another participant:

```
# A tibble: 11 × 4
   out_iresp out_tref out_tresp out_tdelta
 1         1     9375      8258      -1117
 2         2    18750     10141      -8609
 3         3    28125     19211      -8914
 4         4    33750     28500      -5250
 5         5    39375     33779      -5596
 6         6    50625     39463     -11162
 7         7    54375     50962      -3413
 8         8    63750     54528      -9222
 9         9    73125     63861      -9264
10        10    76875     73310      -3565
11        11       -1     76974      76975
```

Here the second click is shifting the whole timeseries

```
# A tibble: 11 × 4
   out_iresp out_tref out_tresp out_tdelta
 1         1     9375      8258      -1117
 2         2    18750     10141      -8609
 3         3    28125     19211      -8914
 4         4    33750     28500      -5250
 5         5    39375     33779      -5596
 6         6    50625     39463     -11162
 7         7    54375     50962      -3413
 8         8    63750     54528      -9222
 9         9    73125     63861      -9264
10        10    76875     73310      -3565
11        11       -1     76974      76975
```

In this case, no extra-click or double click are observed.

We can adopt a realignment algorithm as NW. The aim of the realignment procedure is not to improve temporal accuracy, but to correct only erroneous target–response associations. Therefore, the algorithm should intervene exclusively when introducing an extra response or a missed target yields a more plausible alignment than the original sequence. Otherwise, temporal deviations—even large ones—should be retained as valid measures of participant performance rather than being corrected by the preprocessing procedure.

Parameterization of the Needleman–Wunsch algorithm for the Anticipatory Planning task

Unlike the tapping paradigm, the Anticipatory Planning task does not require continuous synchronization with a periodic beat. Instead, participants are instructed to produce a single response for each target event (i.e., the last note of each rhythmic cluster). Consequently, the purpose of the Needleman–Wunsch algorithm (see other supplementary files for a more detailed explanation) is not to estimate synchronization accuracy but to reconstruct the most plausible correspondence between the sequence of target events and the sequence of participants' responses.

Following visual inspection of individual time series, the alignment procedure was designed to be as conservative as possible. The algorithm should modify the original pairing only when the hypothesis of an insertion (extra response) or deletion (missed response) explains the observed sequence substantially better than the original alignment. Large temporal errors alone should not trigger realignment.

Accordingly, the three parameters of the algorithm are defined with reference to two distinct temporal scales.

1. Double-click threshold

The double-click threshold is based on the temporal extent of a single rhythmic cluster rather than on the underlying beat. Two responses occurring within the temporal boundaries of the same cluster are considered candidate autocorrections (or repeated attempts to respond to the same target) and therefore should not automatically induce a shift of the subsequent alignment.

2. Maximum matching distance

The maximum admissible distance for matching a response to a target is expressed as a proportion of the temporal interval separating consecutive target events. This parameter defines the temporal region within which a response can plausibly be assigned to a given target before it becomes more likely to belong to an adjacent target.

3. Gap penalty

The cost associated with inserting a gap (i.e., declaring an extra response or a missed target) is likewise expressed as a proportion of the interval between consecutive targets. The gap penalty therefore determines the amount of temporal mismatch that the algorithm is willing to tolerate before preferring to realign the sequences through an insertion or deletion.

Overall, this parameterization aims to preserve genuine timing variability while correcting only those cases in which sequence misalignment is substantially more plausible than large temporal error.

Since target events are separated by long temporal intervals, the gap penalty was expressed as a proportion of the interval between consecutive targets. This allows the algorithm to tolerate substantial timing variability while preventing isolated extra responses from propagating alignment errors across the remainder of the sequence.

ANTICIPATORY PLANNING TASK:

Order-preserving temporal realignment

```
############################################################
# ANTICIPATORY PLANNING TASK:
# ORDER-PRESERVING TEMPORAL REALIGNMENT
#
# Assumptions:
# - the input dataset already contains only the TEST condition
# - out_tref  = expected target times
# - out_tresp = recorded response times
# - -1 indicates that no target/response is present on that row
#
# Outputs:
# - per_target
# - false_alarms
# - double_clicks_removed
# - response_status
# - diagnostics
# - dataANT_aligned
############################################################


#LOAD PACKAGES
```

```r
library(dplyr)
library(tibble)
```

```
Attaching package: 'tibble'

The following object is masked from 'package:sjmisc':

    add_case
```

```r
# 1) CENTRAL PARAMETERS

params_ANT <- list(

  # Mechanical double clicks:
  # consecutive responses separated by <= 100 ms
  # are treated as duplicate motor events.
  double_click_ms = 100,

  # A response can be matched to a target only when its
  # temporal distance does not exceed 50% of the relevant
  # local target-to-target interval.
  max_match_frac = 0.50,

  # Cost of declaring one missed target or one false alarm.
  # Match costs are normalised by the relevant local
  # target-to-target interval.
  gap_penalty = 0.50
)



# 2) DATA PREPARATION

# Use a stable identifier to reconnect the aligned output
# to the original dataset.

if (!"rowid" %in% names(dataANT_test)) {

  dataANT_test <- dataANT_test %>%
    mutate(rowid = dplyr::row_number())
}



# 3) REMOVE MECHANICAL DOUBLE CLICKS

# Only extremely close responses are collapsed here.
#
# The first response is retained.
# Subsequent responses within double_click_ms are stored
# separately as double_click_removed.
```

```r
#
# Responses farther apart are NOT removed:
# NW will determine whether they are matched responses
# or false alarms.

collapse_double_clicks_ANT <- function(
    t_resp,
    resp_rowid,
    double_click_ms = 100
) {

  if (length(t_resp) != length(resp_rowid)) {
    stop("t_resp and resp_rowid must have the same length.")
  }

  response_tbl <- tibble(
    t_resp = t_resp,
    resp_rowid = resp_rowid
  ) %>%
    filter(
      !is.na(t_resp),
      is.finite(t_resp),
      t_resp != -1
    ) %>%
    arrange(t_resp, resp_rowid)

  empty_removed <- tibble(
    retained_resp_rowid = integer(0),
    retained_tresp = numeric(0),
    removed_resp_rowid = integer(0),
    removed_tresp = numeric(0),
    interval_ms = numeric(0),
    response_status = character(0)
  )

  if (nrow(response_tbl) <= 1) {

    return(
      list(
        responses = response_tbl,
        removed = empty_removed
      )
    )
  }

  kept_indices <- integer(0)
  removed <- empty_removed

  i <- 1
```

```
  while (i <= nrow(response_tbl)) {

    j <- i

    # Extend the cluster while consecutive responses
    # remain within the mechanical-double-click threshold.
    while (
      j + 1 <= nrow(response_tbl) &&
      (
        response_tbl$t_resp[j + 1] -
          response_tbl$t_resp[j]
      ) <= double_click_ms
    ) {

      j <- j + 1
    }

    cluster_indices <- i:j

    # Keep the first response in the cluster.
    kept_indices <- c(kept_indices, i)

    if (length(cluster_indices) > 1) {

      removed_indices <- cluster_indices[-1]

      removed <- bind_rows(
        removed,
        tibble(
          retained_resp_rowid =
            response_tbl$resp_rowid[i],

          retained_tresp =
            response_tbl$t_resp[i],

          removed_resp_rowid =
            response_tbl$resp_rowid[removed_indices],

          removed_tresp =
            response_tbl$t_resp[removed_indices],

          interval_ms =
            response_tbl$t_resp[removed_indices] -
            response_tbl$t_resp[i],

          response_status =
            "double_click_removed"
        )
      )
```

```r
    }

    i <- j + 1
  }

  list(
    responses = response_tbl[kept_indices, ],
    removed = removed
  )
}



# 4) LOCAL TEMPORAL SCALE FOR EACH TARGET

# For each target:
#
# - an early response is scaled relative to the interval
#   between the current and previous target;
#
# - a late response is scaled relative to the interval
#   between the current and following target.
#
# For the first and final targets, the only available
# adjacent interval is used.

calculate_local_target_intervals <- function(t_ref) {

  n <- length(t_ref)

  if (n < 2) {
    stop(
      "At least two target times are required to calculate ",
      "local target-to-target intervals."
    )
  }

  target_intervals <- diff(t_ref)

  if (
    any(!is.finite(target_intervals)) ||
    any(target_intervals <= 0)
  ) {
    stop(
      "Target times must be finite and strictly increasing."
    )
  }

  previous_interval <- numeric(n)
  next_interval <- numeric(n)
```

```r
  # First target: use the following interval on both sides.
  previous_interval[1] <- target_intervals[1]
  next_interval[1] <- target_intervals[1]

  # Last target: use the preceding interval on both sides.
  previous_interval[n] <- target_intervals[n - 1]
  next_interval[n] <- target_intervals[n - 1]

  if (n > 2) {

    previous_interval[2:(n - 1)] <-
      target_intervals[1:(n - 2)]

    next_interval[2:(n - 1)] <-
      target_intervals[2:(n - 1)]
  }

  tibble(
    target_index = seq_len(n),
    previous_target_interval_ms = previous_interval,
    next_target_interval_ms = next_interval
  )
}



# 5) TARGET–RESPONSE MATCH COST

# The match cost is:
#
# |response - target| / relevant local target interval
#
# Thus:
#
# 0.00 = exact response
# 0.25 = error equal to 25% of the local interval
# 0.50 = error equal to 50% of the local interval
#
# A match is forbidden if the normalised distance exceeds
# max_match_frac.

local_match_cost <- function(
    target_time,
    response_time,
    previous_interval_ms,
    next_interval_ms,
    max_match_frac
) {

  delta_ms <- response_time - target_time
```

```r
  # Anticipated response:
  # scale against the distance from the previous target.
  if (delta_ms < 0) {

    local_interval_ms <- previous_interval_ms

  } else {

    # Delayed response:
    # scale against the distance to the following target.
    local_interval_ms <- next_interval_ms
  }

  if (
    !is.finite(local_interval_ms) ||
    local_interval_ms <= 0
  ) {
    return(Inf)
  }

  normalised_distance <-
    abs(delta_ms) / local_interval_ms

  if (normalised_distance > max_match_frac) {
    return(Inf)
  }

  normalised_distance
}



# 6) NEEDLEMAN–WUNSCH DYNAMIC-PROGRAMMING ALIGNMENT

align_times_dp_ANT <- function(
    t_ref,
    t_resp,
    previous_intervals_ms,
    next_intervals_ms,
    gap_penalty = 0.50,
    max_match_frac = 0.50
) {

  n <- length(t_ref)
  m <- length(t_resp)

  if (
    length(previous_intervals_ms) != n ||
    length(next_intervals_ms) != n
```

```
) {
  stop(
    "The local-interval vectors must have one value ",
    "for every target."
  )
}

# Cumulative-cost matrix.
C <- matrix(
  Inf,
  nrow = n + 1,
  ncol = m + 1
)

# Backtracking matrix:
#
# 1 = match
# 2 = missed target
# 3 = false alarm
B <- matrix(
  NA_integer_,
  nrow = n + 1,
  ncol = m + 1
)

C[1, 1] <- 0

# First column:
# all target events are missed.
if (n > 0) {

  for (i in seq_len(n)) {

    C[i + 1, 1] <-
      C[i, 1] + gap_penalty

    B[i + 1, 1] <- 2L
  }
}

# First row:
# all responses are false alarms.
if (m > 0) {

  for (j in seq_len(m)) {

    C[1, j + 1] <-
      C[1, j] + gap_penalty

    B[1, j + 1] <- 3L
```

```
  }
}

# Fill the dynamic-programming matrices.
if (n > 0 && m > 0) {

  for (i in seq_len(n)) {

    for (j in seq_len(m)) {

      match_cost <- local_match_cost(
        target_time = t_ref[i],
        response_time = t_resp[j],
        previous_interval_ms =
          previous_intervals_ms[i],
        next_interval_ms =
          next_intervals_ms[i],
        max_match_frac = max_match_frac
      )

      candidate_costs <- c(

        # Target-response match.
        C[i, j] + match_cost,

        # Missed target.
        C[i, j + 1] + gap_penalty,

        # False alarm.
        C[i + 1, j] + gap_penalty
      )

      best_move <- which.min(candidate_costs)

      C[i + 1, j + 1] <-
        candidate_costs[best_move]

      B[i + 1, j + 1] <-
        best_move
    }
  }
}

# Backtracking.
i <- n + 1
j <- m + 1

matches <- list()
missed_targets <- integer(0)
```

```
false_alarms <- integer(0)

while (i > 1 || j > 1) {

  move <- B[i, j]

  if (is.na(move)) {

    stop(
      paste0(
        "Undefined backtracking move at matrix position i = ",
        i,
        ", j = ",
        j,
        "."
      )
    )
  }

  if (move == 1L) {

    matches[[length(matches) + 1]] <-
      c(i - 1, j - 1)

    i <- i - 1
    j <- j - 1

  } else if (move == 2L) {

    missed_targets <- c(
      missed_targets,
      i - 1
    )

    i <- i - 1

  } else {

    false_alarms <- c(
      false_alarms,
      j - 1
    )

    j <- j - 1
  }
}

matches <- if (length(matches) > 0) {
```

```
    do.call(
      rbind,
      rev(matches)
    )

  } else {

    NULL
  }

  list(
    matches = matches,
    missed_targets = sort(missed_targets),
    false_alarms = sort(false_alarms),
    total_cost = C[n + 1, m + 1]
  )
}


# 7) REALIGN ONE PARTICIPANT/STIMULUS SEQUENCE

realign_one_sequence_ANT <- function(
    df_sequence,
    p = params_ANT
) {

  required_columns <- c(
    "rowid",
    "out_tref",
    "out_tresp"
  )

  missing_columns <- setdiff(
    required_columns,
    names(df_sequence)
  )

  if (length(missing_columns) > 0) {

    stop(
      paste(
        "Missing required columns:",
        paste(missing_columns, collapse = ", ")
      )
    )
  }

  # ----------------------------------------------------------
  # Reconstruct the target sequence independently.
  #
```

```r
# A row with out_tref == -1 is not a target row, but its
# out_tresp may still contain a genuine response.
# ----------------------------------------------------------

target_tbl <- df_sequence %>%
  filter(
    !is.na(out_tref),
    is.finite(out_tref),
    out_tref != -1
  ) %>%
  transmute(
    target_rowid = rowid,
    t_ref = out_tref
  ) %>%
  arrange(t_ref, target_rowid)

# ----------------------------------------------------------
# Reconstruct the response sequence independently.
#
# A row with out_tresp == -1 contains no response, but its
# out_tref may still represent a missed target.
# ----------------------------------------------------------

response_tbl_raw <- df_sequence %>%
  filter(
    !is.na(out_tresp),
    is.finite(out_tresp),
    out_tresp != -1
  ) %>%
  transmute(
    resp_rowid = rowid,
    t_resp = out_tresp
  ) %>%
  arrange(t_resp, resp_rowid)

if (nrow(target_tbl) < 2) {

  stop(
    "This sequence contains fewer than two valid targets."
  )
}

# Collapse only mechanical double clicks.
collapsed <- collapse_double_clicks_ANT(
  t_resp = response_tbl_raw$t_resp,
  resp_rowid = response_tbl_raw$resp_rowid,
  double_click_ms = p$double_click_ms
)

response_tbl_clean <- collapsed$responses
```

```r
double_clicks_removed <- collapsed$removed

# Calculate target-specific temporal scales.
local_intervals <- calculate_local_target_intervals(
  target_tbl$t_ref
)

target_tbl <- target_tbl %>%
  bind_cols(
    local_intervals %>%
      dplyr::select(
        previous_target_interval_ms,
        next_target_interval_ms
      )
  )

# Run NW alignment.
alignment <- align_times_dp_ANT(
  t_ref = target_tbl$t_ref,
  t_resp = response_tbl_clean$t_resp,
  previous_intervals_ms =
    target_tbl$previous_target_interval_ms,
  next_intervals_ms =
    target_tbl$next_target_interval_ms,
  gap_penalty = p$gap_penalty,
  max_match_frac = p$max_match_frac
)

# ------------------------------------------------------------
# Target-level output
# ------------------------------------------------------------

per_target <- target_tbl %>%
  mutate(
    target_status = "matched",
    matched_resp_index = NA_integer_,
    matched_resp_rowid = NA_integer_,
    t_resp_aligned = NA_real_,
    out_tdelta_aligned = NA_real_,
    local_interval_used_ms = NA_real_,
    normalised_match_cost = NA_real_
  )

if (
  !is.null(alignment$matches) &&
  nrow(alignment$matches) > 0
) {

  matched_target_indices <-
    alignment$matches[, 1]
```

```
  matched_response_indices <-
    alignment$matches[, 2]

  per_target$matched_resp_index[
    matched_target_indices
  ] <- matched_response_indices

  per_target$matched_resp_rowid[
    matched_target_indices
  ] <- response_tbl_clean$resp_rowid[
    matched_response_indices
  ]

  per_target$t_resp_aligned[
    matched_target_indices
  ] <- response_tbl_clean$t_resp[
    matched_response_indices
  ]

  per_target$out_tdelta_aligned[
    matched_target_indices
  ] <-
    per_target$t_resp_aligned[
      matched_target_indices
    ] -
    per_target$t_ref[
      matched_target_indices
    ]

  # Store the local denominator actually used.
  for (k in matched_target_indices) {

    if (per_target$out_tdelta_aligned[k] < 0) {

      per_target$local_interval_used_ms[k] <-
        per_target$previous_target_interval_ms[k]

    } else {

      per_target$local_interval_used_ms[k] <-
        per_target$next_target_interval_ms[k]
    }

    per_target$normalised_match_cost[k] <-
      abs(per_target$out_tdelta_aligned[k]) /
      per_target$local_interval_used_ms[k]
  }
}
```

```r
# Missed targets.
if (length(alignment$missed_targets) > 0) {

  per_target$target_status[
    alignment$missed_targets
  ] <- "missed_target"
}

# ------------------------------------------------------------
# False-alarm output
# ------------------------------------------------------------

false_alarms <- if (
  length(alignment$false_alarms) > 0
) {

  response_tbl_clean[
    alignment$false_alarms,
    ,
    drop = FALSE
  ] %>%
    mutate(
      response_status = "false_alarm"
    )

} else {

  tibble(
    resp_rowid = integer(0),
    t_resp = numeric(0),
    response_status = character(0)
  )
}

# ------------------------------------------------------------
# Response-level status
# ------------------------------------------------------------

response_status <- response_tbl_clean %>%
  mutate(
    response_status = "matched_response",
    matched_target_rowid = NA_integer_
  )

if (
  !is.null(alignment$matches) &&
  nrow(alignment$matches) > 0
) {
```

```
  matched_target_indices <-
    alignment$matches[, 1]

  matched_response_indices <-
    alignment$matches[, 2]

  response_status$matched_target_rowid[
    matched_response_indices
  ] <- target_tbl$target_rowid[
    matched_target_indices
  ]
}

if (length(alignment$false_alarms) > 0) {

  response_status$response_status[
    alignment$false_alarms
  ] <- "false_alarm"
}

# Add mechanically removed responses.
if (nrow(double_clicks_removed) > 0) {

  double_click_status <-
    double_clicks_removed %>%
    transmute(
      resp_rowid = removed_resp_rowid,
      t_resp = removed_tresp,
      response_status =
        "double_click_removed",
      matched_target_rowid = NA_integer_
    )

  response_status <- bind_rows(
    response_status,
    double_click_status
  ) %>%
    arrange(t_resp, resp_rowid)
}

# ------------------------------------------------------------
# Diagnostics
# ------------------------------------------------------------

diagnostics <- tibble(
  total_alignment_cost =
    alignment$total_cost,

  n_targets =
    nrow(target_tbl),
```

```
    n_raw_responses =
      nrow(response_tbl_raw),

    n_clean_responses =
      nrow(response_tbl_clean),

    n_matches =
      sum(per_target$target_status == "matched"),

    n_missed_targets =
      sum(per_target$target_status == "missed_target"),

    n_false_alarms =
      nrow(false_alarms),

    n_double_clicks_removed =
      nrow(double_clicks_removed),

    double_click_ms =
      p$double_click_ms,

    max_match_frac =
      p$max_match_frac,

    gap_penalty =
      p$gap_penalty
  )

  list(
    per_target = per_target,
    false_alarms = false_alarms,
    double_clicks_removed =
      double_clicks_removed,
    response_status = response_status,
    diagnostics = diagnostics
  )
}


# 8) REALIGN THE COMPLETE DATASET


realign_all_ANT <- function(
    data,
    group_vars = c("participant_ID", "id_audio"),
    p = params_ANT
) {
```

```r
missing_group_vars <- setdiff(
  group_vars,
  names(data)
)

if (length(missing_group_vars) > 0) {

  stop(
    paste(
      "Missing grouping columns:",
      paste(missing_group_vars, collapse = ", ")
    )
  )
}

keys <- data %>%
  distinct(
    across(all_of(group_vars))
  ) %>%
  arrange(
    across(all_of(group_vars))
  )

per_target_all <- list()
false_alarms_all <- list()
double_clicks_all <- list()
response_status_all <- list()
diagnostics_all <- list()

for (i in seq_len(nrow(keys))) {

  current_key <- keys[i, , drop = FALSE]

  df_sequence <- data

  for (group_variable in group_vars) {

    current_value <-
      current_key[[group_variable]][1]

    if (is.na(current_value)) {

      df_sequence <- df_sequence %>%
        filter(is.na(.data[[group_variable]]))

    } else {

      df_sequence <- df_sequence %>%
        filter(
```

```
            .data[[group_variable]] ==
              current_value
          )
      }
    }

    if (nrow(df_sequence) == 0) {
      next
    }

    result <- realign_one_sequence_ANT(
      df_sequence = df_sequence,
      p = p
    )

    add_group_values <- function(tbl) {

      if (is.null(tbl)) {
        return(tbl)
      }

      for (group_variable in group_vars) {

        tbl[[group_variable]] <-
          current_key[[group_variable]][1]
      }

      tbl
    }

    per_target_all[[length(per_target_all) + 1]] <-
      add_group_values(result$per_target)

    false_alarms_all[[length(false_alarms_all) + 1]] <-
add_group_values(result$false_alarms)

    double_clicks_all[[length(double_clicks_all) + 1]] <- add_group_values(
      result$double_clicks_removed
    )

    response_status_all[[length(response_status_all) + 1]] <- add_group_values(
      result$response_status
    )

    diagnostics_all[[length(diagnostics_all) + 1]] <- add_group_values(
      result$diagnostics
    )
  }
```

```
  list(

    per_target =
      bind_rows(per_target_all),

    false_alarms =
      bind_rows(false_alarms_all),

    double_clicks_removed =
      bind_rows(double_clicks_all),

    response_status =
      bind_rows(response_status_all),

    diagnostics =
      bind_rows(diagnostics_all)
  )
}


# 9) RUN THE COMPLETE REALIGNMENT

RA_ANT <- realign_all_ANT(
  data = dataANT_test,
  group_vars = c("participant_ID", "id_audio"),
  p = params_ANT
)


# 10) CREATE OUTPUT OBJECTS

per_target <- RA_ANT$per_target %>%
  mutate(alignment_version = "central")

false_alarms <- RA_ANT$false_alarms %>%
  mutate(alignment_version = "central")

double_clicks_removed <-
  RA_ANT$double_clicks_removed %>%
  mutate(alignment_version = "central")

response_status <- RA_ANT$response_status %>%
  mutate(alignment_version = "central")

diagnostics <- RA_ANT$diagnostics %>%
  mutate(alignment_version = "central")


# 11) JOIN ALIGNED INFORMATION BACK TO THE ORIGINAL DATASET
```

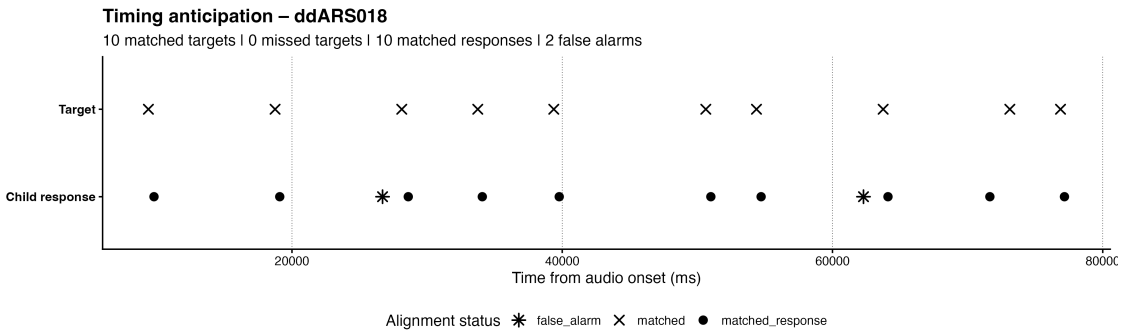
```r
dataANT_aligned <- dataANT_test %>%

  # Target-level alignment:
  left_join(
    per_target %>%
      dplyr::select(
        target_rowid,
        target_status,
        matched_resp_rowid,
        t_resp_aligned,
        out_tdelta_aligned,
        previous_target_interval_ms,
        next_target_interval_ms,
        local_interval_used_ms,
        normalised_match_cost
      ),
    by = c("rowid" = "target_rowid")
  ) %>%

  # Status of every recorded response:
  left_join(
    response_status %>%
      dplyr::select(
        resp_rowid,
        response_status,
        matched_target_rowid
      ),
    by = c("rowid" = "resp_rowid")
  ) %>%

  mutate(
    alignment_version = "central",
    double_click_ms =
      params_ANT$double_click_ms,
    max_match_frac =
      params_ANT$max_match_frac,
    gap_penalty =
      params_ANT$gap_penalty
  )


# 12) CHECK THE OUTPUTS

View(per_target)

View(false_alarms)

View(double_clicks_removed)
```

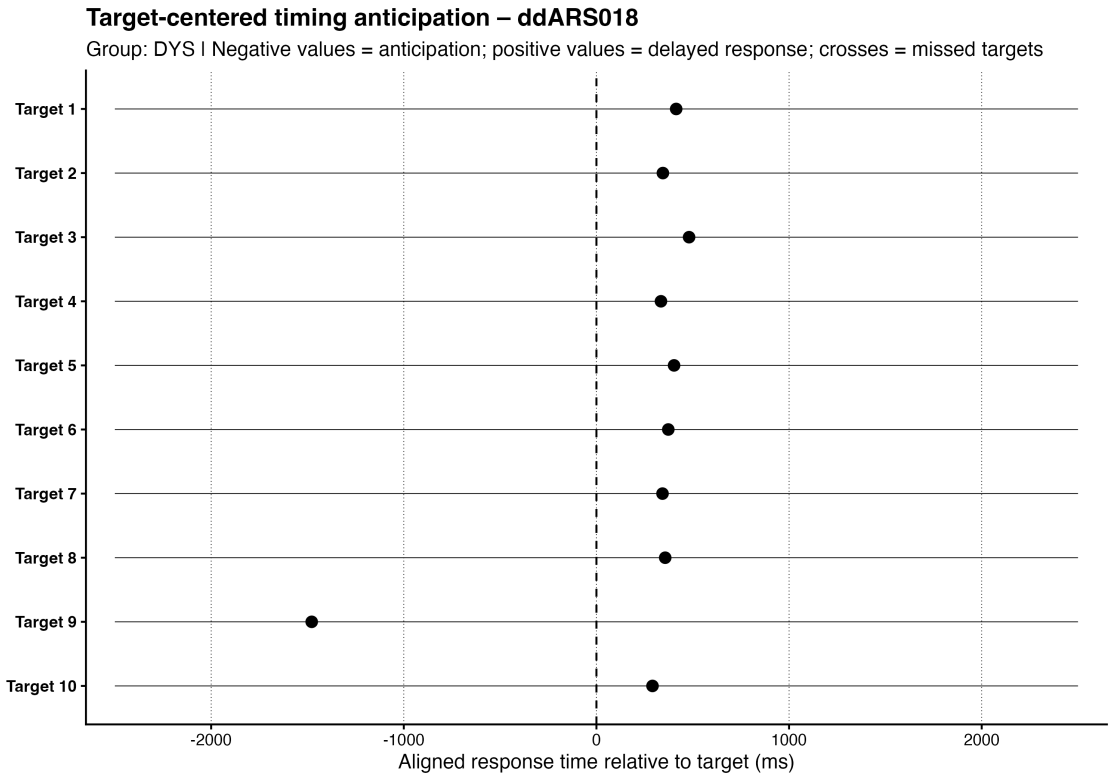
```
View(response_status)

View(diagnostics)

View(dataANT_aligned)
```

In contrast to the tapping paradigm, all temporal thresholds were defined relative to the local interval between consecutive target events rather than to a fixed metrical period. Specifically, the maximum admissible target–response distance (`max_match_frac`) and the gap penalty were both set to **0.50**, meaning that responses could only be matched when their temporal deviation did not exceed 50% of the adjacent target-to-target interval. Likewise, the cost of declaring a missed target or a false alarm was set to **0.50** in normalized units, corresponding to the temporal error associated with half of the local target interval. Mechanical double clicks were identified separately using a fixed threshold of **100 ms**, reflecting rapid duplicate motor responses rather than genuine attempts to respond to different target events.

Plots realignment for participant level

The case in which a double click shifted the timeseries, is now solved:






```
# A tibble: 10 × 8
   out_iresp out_tref out_tresp t_resp_aligned out_tdelta out_tdelta_aligned
 1         1     9375      9789           9789        414                414
 2         2    18750     19095          19095        345                345
 3         3    28125     26703          28606      -1422                481
 4         4    33750     28606          34085      -5144                335
 5         5    39375     34085          39778      -5290                403
 6         6    50625     39778          50998     -10847                373
 7         7    54375     50998          54718      -3377                343
 8         8    63750     54718          64107      -9032                357
 9         9    73125     62294          71647     -10831              -1478
10        10    76875     64107          77166     -12768                291
```

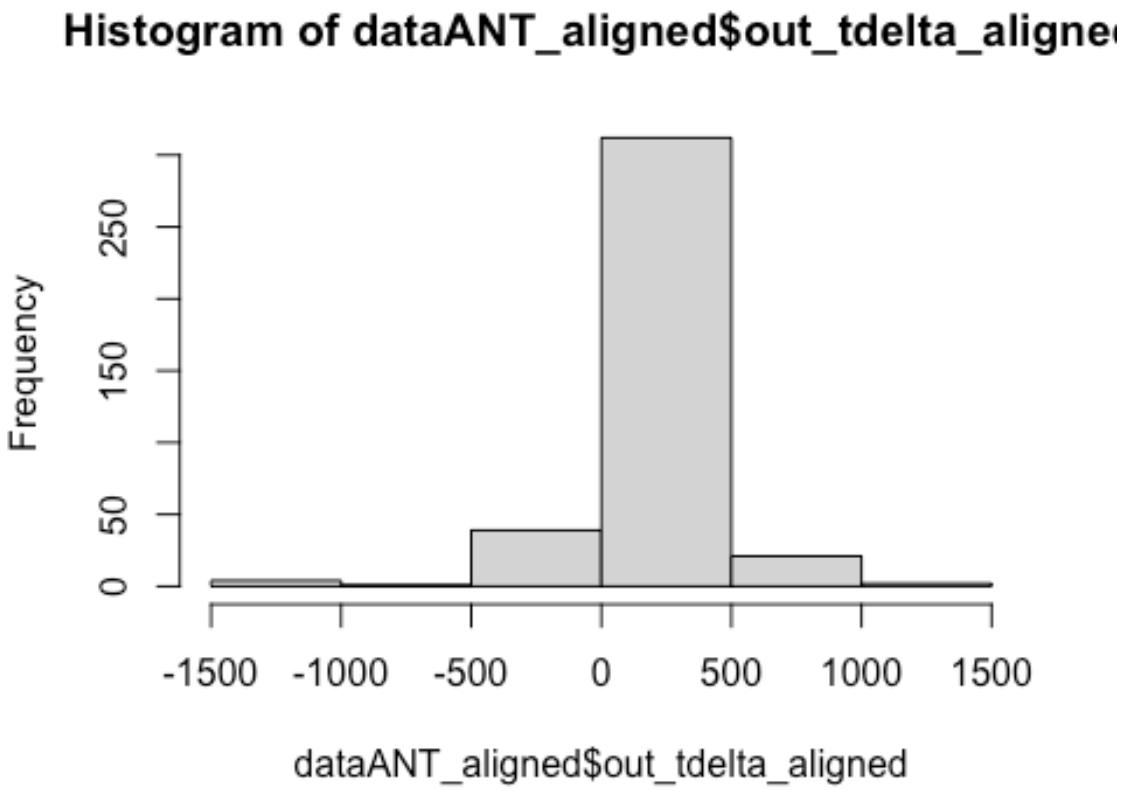
Also in this case, in whic an additive click shifted the timeseries, all the clicks are now realigned to the target:

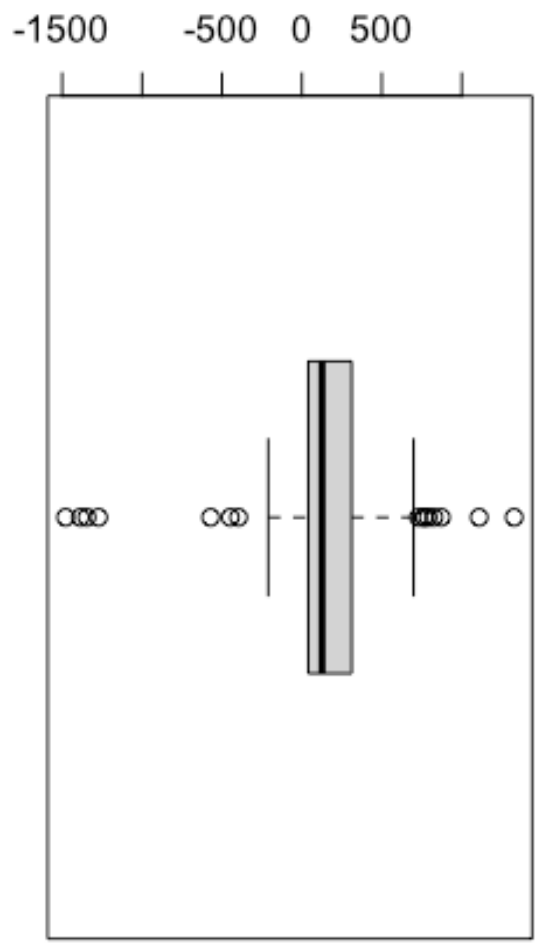
```
# A tibble: 10 × 8
   out_iresp out_tref out_tresp t_resp_aligned out_tdelta out_tdelta_aligned
 1         1     9375      8258          10141      -1117                766
 2         2    18750     10141          19211      -8609                461
 3         3    28125     19211          28500      -8914                375
 4         4    33750     28500          33779      -5250                 29
 5         5    39375     33779          39463      -5596                 88
 6         6    50625     39463          50962     -11162                337
 7         7    54375     50962          54528      -3413                153
 8         8    63750     54528          63861      -9222                111
 9         9    73125     63861          73310      -9264                185
10        10    76875     73310          76974      -3565                 99
```

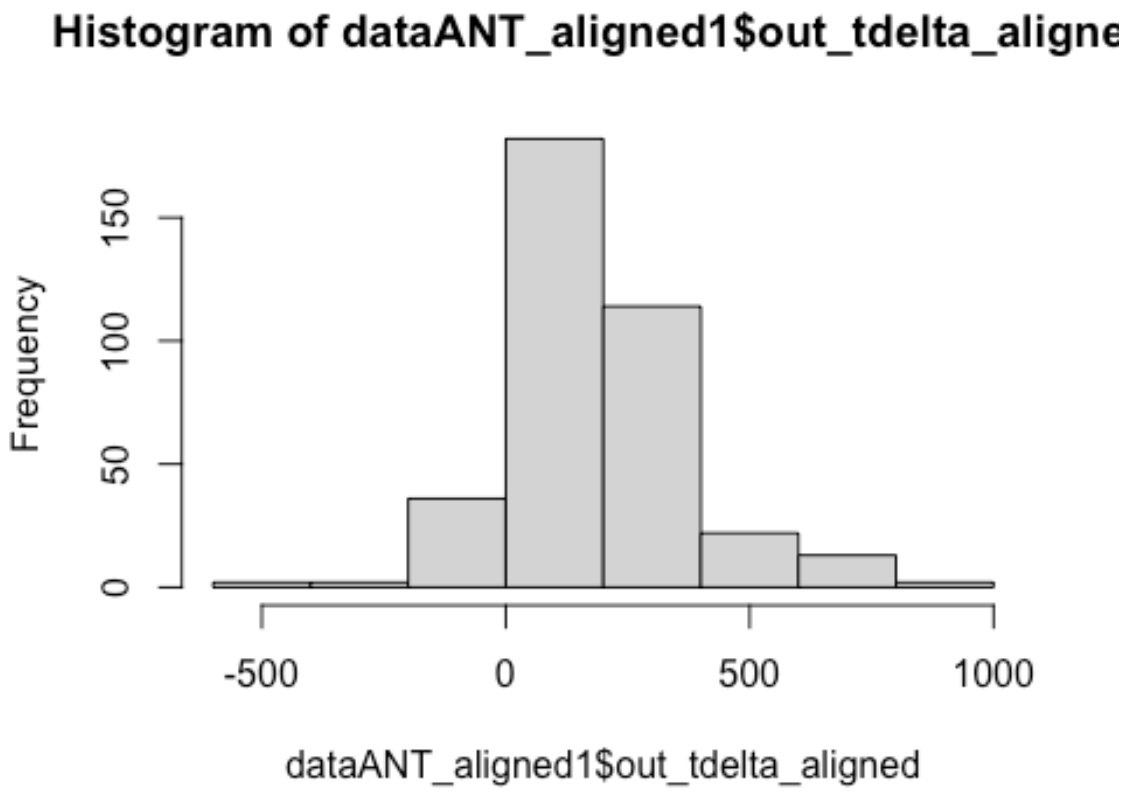
GROUP COMPARISON

Statistical model

```
                       DYS TDC
  double_click_removed   0   2
  false_alarm           20  14
  matched_response     185 194


    Fisher's Exact Test for Count Data

data:  table(dataANT_aligned$response_status, dataANT_aligned$Group)
p-value = 0.196
alternative hypothesis: two.sided
```

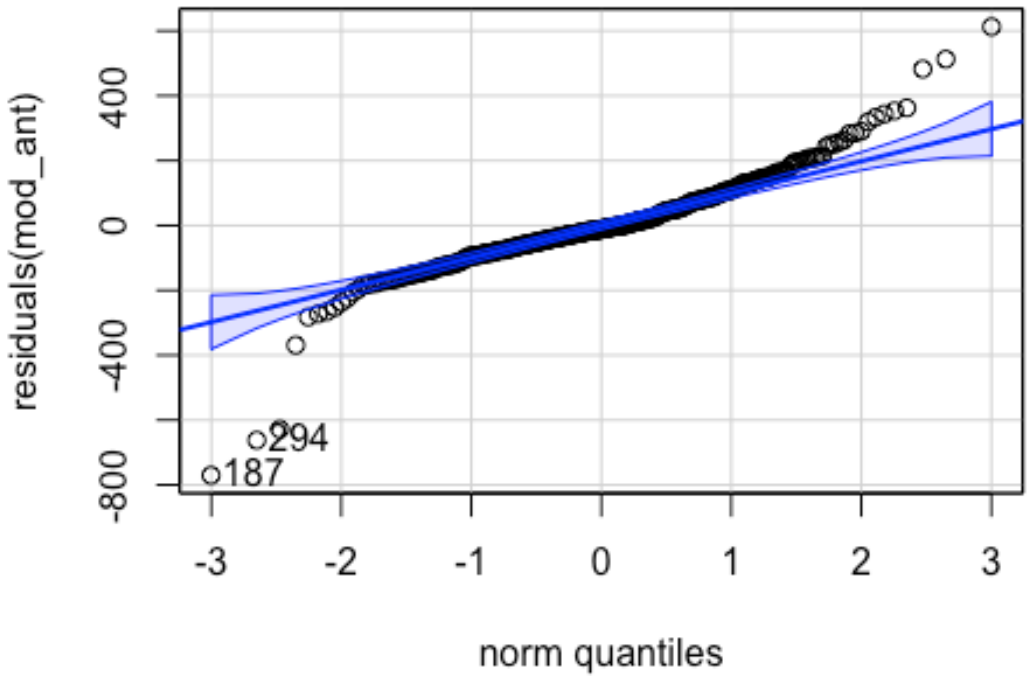



-1500
-500
0
500




```
Analysis of Deviance Table (Type III Wald chisquare tests)

Response: out_tdelta_aligned
              Chisq Df Pr(>Chisq)
(Intercept) 36.2992  1  1.692e-09 ***
Group        0.3012  1     0.5831
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Linear mixed model fit by REML ['lmerMod']
Formula: out_tdelta_aligned ~ Group + (1 | participant_ID)
   Data: dataANT_aligned1

REML criterion at convergence: 4823

Scaled residuals:
    Min      1Q  Median      3Q     Max
-5.4173 -0.4719 -0.0784  0.4692  4.3076

Random effects:
 Groups         Name        Variance Std.Dev.
 participant_ID (Intercept) 17026    130.5
 Residual                   20223    142.2
Number of obs: 373, groups:  participant_ID, 39
```

```
Fixed effects:
            Estimate Std. Error t value
(Intercept)   191.52      31.79   6.025
GroupTDC      -24.34      44.35  -0.549

Correlation of Fixed Effects:
         (Intr)
GroupTDC -0.717
```




```
   Accurate responses:


    DYS TDC
  0  86  75
  1  99 119

# A tibble: 33 × 4
   Group participant_ID accuratezza Count
 1 DYS   ddARS002       1                6
 2 DYS   ddARS003       1                8
 3 DYS   ddARS006       1                1
 4 DYS   ddARS007       1                1
 5 DYS   ddARS008       1                7
 6 DYS   ddARS009       1                7
```

```
 7 DYS   ddARS010       1               9
 8 DYS   ddARS011       1               6
 9 DYS   ddARS012       1               6
10 DYS   ddARS013       1               7
```

Percentage of accurate responses on the whole sample:

```
 0        1
42.48% 57.51%
```

Group-comparison on accuracy

We will run a non-parametric group comparison on participant's total score

```
    Wilcoxon rank sum test with continuity correction

data:  count_accTEST$Ant_acc by count_accTEST$Group
W = 104.5, p-value = 0.2578
alternative hypothesis: true location shift is not equal to 0
```