

DPLYR Bisection—Evaluate Many Unknown Nonlinear Equations Jointly, Solve Roots for Strictly Monotonic Functions with Single Zero-Crossing

Fan Wang

Contents

Bisection	1
---------------------	---

Bisection

Go to the [RMD](#), [R](#), [PDF](#), or [HTML](#) version of this file. Go back to [fan's REconTools Package](#), [R4Econ Repository \(bookdown site\)](#), or [Intro Stats with R Repository](#).

See the `ff_opti_bisect_pmap_multi` function from [Fan's REconTools Package](#), which provides a reusable function based on the algorithm worked out here.

The bisection specific code does not need to do much.

- list variables in file for grouping, each group is an individual for whom we want to calculate optimal choice for using bisection.
- string variable name of input where functions are evaluated, these are already contained in the dataframe, existing variable names, row specific, rowwise computation over these, each rowwise calculation using different rows.
- scalar and array values that are applied to every rowwise calculation, all rowwise calculations using the same scalars and arrays.
- string output variable name

This is how I implement the bisection algorithm, when we know the bounding minimum and maximum to be below and above zero already.

1. Evaluate $f_a^0 = f(a^0)$ and $f_b^0 = f(b^0)$, min and max points.
2. Evaluate at $f_p^0 = f(p^0)$, where $p_0 = \frac{a^0 + b^0}{2}$.
3. if $f_a^i \cdot f_p^i < 0$, then $b_{i+1} = p_i$, else, $a_{i+1} = p_i$ and $f_a^{i+1} = p_i$.
4. iterate until convergence.

Generate New columns of a and b as we iterate, do not need to store p, p is temporary. Evaluate the function below which we have already tested, but now, in the dataframe before generating all permutations, `tb_states_choices`, now the `fl_N` element will be changing with each iteration, it will be row specific. `fl_N` are first min and max, then each subsequent ps.

Initialize Matrix First, initialize the matrix with a_0 and b_0 , the initial min and max points:

```
# common prefix to make reshaping easier
st_bisec_prefix <- 'bisec_'
svr_a_lst <- paste0(st_bisec_prefix, 'a_0')
svr_b_lst <- paste0(st_bisec_prefix, 'b_0')
svr_fa_lst <- paste0(st_bisec_prefix, 'fa_0')
svr_fb_lst <- paste0(st_bisec_prefix, 'fb_0')
```

```

# Add initial a and b
tb_states_choices_bisec <- tb_states_choices %>%
  mutate(!sym(svr_a_lst) := fl_N_min, !sym(svr_b_lst) := fl_N_agg)

# Evaluate function f(a_0) and f(b_0)
tb_states_choices_bisec <- tb_states_choices_bisec %>%
  rowwise() %>%
  mutate(!sym(svr_fa_lst) := ffi_nonlin_dplyrdo(fl_A, fl_alpha, !sym(svr_a_lst),
                                                ar_nN_A, ar_nN_alpha,
                                                fl_N_agg, fl_rho),
         !sym(svr_fb_lst) := ffi_nonlin_dplyrdo(fl_A, fl_alpha, !sym(svr_b_lst),
                                                ar_nN_A, ar_nN_alpha,
                                                fl_N_agg, fl_rho))

# Summarize
dim(tb_states_choices_bisec)

```

```
## [1] 4 7
```

```
summary(tb_states_choices_bisec)
```

##	INDI_ID	fl_A	fl_alpha	bisec_a_0	bisec_b_0	bisec_fa_0	bisec_fb_0
##	Min. :1.00	Min. :-2	Min. :0.1	Min. :0	Min. :100	Min. :100	Min. :-45152754
##	1st Qu.:1.75	1st Qu.: -1	1st Qu.:0.3	1st Qu.:0	1st Qu.:100	1st Qu.:100	1st Qu.: -12853672
##	Median :2.50	Median : 0	Median :0.5	Median :0	Median :100	Median :100	Median : -1102920
##	Mean :2.50	Mean : 0	Mean :0.5	Mean :0	Mean :100	Mean :100	Mean : -11843015
##	3rd Qu.:3.25	3rd Qu.: 1	3rd Qu.:0.7	3rd Qu.:0	3rd Qu.:100	3rd Qu.:100	3rd Qu.: -92263
##	Max. :4.00	Max. : 2	Max. :0.9	Max. :0	Max. :100	Max. :100	Max. : -13465

Iterate and Solve for f(p), update f(a) and f(b) Implement the DPLYR based Concurrent bisection algorithm.

```

# fl_tol = float tolerance criteria
# it_tol = number of iterations to allow at most
fl_tol <- 10^-2
it_tol <- 100

# fl_p_dist2zr = distance to zero to initialize
fl_p_dist2zr <- 1000
it_cur <- 0
while (it_cur <= it_tol && fl_p_dist2zr >= fl_tol ) {

  it_cur <- it_cur + 1

  # New Variables
  svr_a_cur <- paste0(st_bisec_prefix, 'a_', it_cur)
  svr_b_cur <- paste0(st_bisec_prefix, 'b_', it_cur)
  svr_fa_cur <- paste0(st_bisec_prefix, 'fa_', it_cur)
  svr_fb_cur <- paste0(st_bisec_prefix, 'fb_', it_cur)

  # Evaluate function f(a_0) and f(b_0)
  # 1. generate p
  # 2. generate f_p
  # 3. generate f_p*f_a
  tb_states_choices_bisec <- tb_states_choices_bisec %>%

```

```

    rowwise() %>%
    mutate(p = ((!!sym(svr_a_lst) + !!sym(svr_b_lst))/2)) %>%
    mutate(f_p = ffi_nonlin_dplyrdo(fl_A, fl_alpha, p,
                                   ar_nN_A, ar_nN_alpha,
                                   fl_N_agg, fl_rho)) %>%
    mutate(f_p_t_f_a = f_p*!!sym(svr_fa_lst))
# fl_p_dist2zr = sum(abs(p))
fl_p_dist2zr <- mean(abs(tb_states_choices_bisec %>% pull(f_p)))

# Update a and b
tb_states_choices_bisec <- tb_states_choices_bisec %>%
  mutate(!!sym(svr_a_cur) :=
    case_when(f_p_t_f_a < 0 ~ !!sym(svr_a_lst),
              TRUE ~ p)) %>%
  mutate(!!sym(svr_b_cur) :=
    case_when(f_p_t_f_a < 0 ~ p,
              TRUE ~ !!sym(svr_b_lst)))
# Update f(a) and f(b)
tb_states_choices_bisec <- tb_states_choices_bisec %>%
  mutate(!!sym(svr_fa_cur) :=
    case_when(f_p_t_f_a < 0 ~ !!sym(svr_fa_lst),
              TRUE ~ f_p)) %>%
  mutate(!!sym(svr_fb_cur) :=
    case_when(f_p_t_f_a < 0 ~ f_p,
              TRUE ~ !!sym(svr_fb_lst)))
# Save from last
svr_a_lst <- svr_a_cur
svr_b_lst <- svr_b_cur
svr_fa_lst <- svr_fa_cur
svr_fb_lst <- svr_fb_cur

# Summar current round
print(paste0('it_cur:', it_cur, ', fl_p_dist2zr:', fl_p_dist2zr))
summary(tb_states_choices_bisec %>%
  select(one_of(svr_a_cur, svr_b_cur, svr_fa_cur, svr_fb_cur)))
}

```

```

## [1] "it_cur:1, fl_p_dist2zr:3671881.19665787"
## [1] "it_cur:2, fl_p_dist2zr:1144985.08219663"
## [1] "it_cur:3, fl_p_dist2zr:359541.34366151"
## [1] "it_cur:4, fl_p_dist2zr:113856.193431704"
## [1] "it_cur:5, fl_p_dist2zr:36406.6254019037"
## [1] "it_cur:6, fl_p_dist2zr:11755.7247291811"
## [1] "it_cur:7, fl_p_dist2zr:3815.91500125466"
## [1] "it_cur:8, fl_p_dist2zr:1229.03892892158"
## [1] "it_cur:9, fl_p_dist2zr:381.513462638575"
## [1] "it_cur:10, fl_p_dist2zr:106.038527344308"
## [1] "it_cur:11, fl_p_dist2zr:31.326905419781"
## [1] "it_cur:12, fl_p_dist2zr:15.6131239505113"
## [1] "it_cur:13, fl_p_dist2zr:3.23620736098339"
## [1] "it_cur:14, fl_p_dist2zr:7.78098110622511"
## [1] "it_cur:15, fl_p_dist2zr:3.44385297666378"
## [1] "it_cur:16, fl_p_dist2zr:2.01882997203239"
## [1] "it_cur:17, fl_p_dist2zr:0.834469221089261"

```

```
## [1] "it_cur:18, fl_p_dist2zr:0.220671530403298"
## [1] "it_cur:19, fl_p_dist2zr:0.0871882680059457"
## [1] "it_cur:20, fl_p_dist2zr:0.125470672506289"
## [1] "it_cur:21, fl_p_dist2zr:0.0521762500154281"
## [1] "it_cur:22, fl_p_dist2zr:0.0308507046075128"
## [1] "it_cur:23, fl_p_dist2zr:0.0127295496732174"
## [1] "it_cur:24, fl_p_dist2zr:0.00345115540382679"
```

Reshape Wide to long to Wide To view results easily, how iterations improved to help us find the roots, convert table from wide to long. Pivot twice. This allows us to easily graph out how bisection is working out iterationby iteration.

Here, we will first show what the raw table looks like, the wide only table, and then show the long version, and finally the version that is medium wide.

Table One-Very Wide Show what the `tb_states_choices_bisec` looks like.

Variables are formatted like: `bisec_xx_yy`, where `yy` is the iteration indicator, and `xx` is either a, b, fa, or fb.

```
head(tb_states_choices_bisec, 10)
```

```
## Source: local data frame [4 x 106]
## Groups: <by row>
##
## # A tibble: 4 x 106
##   INDI_ID  fl_A fl_alpha bisec_a_0 bisec_b_0 bisec_fa_0 bisec_fb_0      p      f_p f_p_t_f_a bisec_
##   <int>  <dbl>  <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  <dbl>  <dbl>    <dbl>    <dbl>
## 1      1    -2    0.1        0      100      100   -13465.  0.480  -1.63e-4  -1.52e-7
## 2      2  -0.667  0.367        0      100      100  -118529.  0.220  -7.00e-4  -1.59e-6
## 3      3   0.667  0.633        0      100      100 -2087311.  0.0821 -2.14e-3  -1.58e-5
## 4      4     2    0.9        0      100      100 -45152754.  0.0364  1.08e-2   3.86e-4
## # ... with 90 more variables: bisec_fa_2 <dbl>, bisec_fb_2 <dbl>, bisec_a_3 <dbl>, bisec_b_3 <dbl>, bisec_fa_3 <dbl>,
## # bisec_b_4 <dbl>, bisec_fa_4 <dbl>, bisec_fb_4 <dbl>, bisec_a_5 <dbl>, bisec_b_5 <dbl>, bisec_fa_5 <dbl>,
## # bisec_b_6 <dbl>, bisec_fa_6 <dbl>, bisec_fb_6 <dbl>, bisec_a_7 <dbl>, bisec_b_7 <dbl>, bisec_fa_7 <dbl>,
## # bisec_b_8 <dbl>, bisec_fa_8 <dbl>, bisec_fb_8 <dbl>, bisec_a_9 <dbl>, bisec_b_9 <dbl>, bisec_fa_9 <dbl>,
## # bisec_b_10 <dbl>, bisec_fa_10 <dbl>, bisec_fb_10 <dbl>, bisec_a_11 <dbl>, bisec_b_11 <dbl>, bisec_fa_11 <dbl>,
## # bisec_b_12 <dbl>, bisec_fa_12 <dbl>, bisec_fb_12 <dbl>, bisec_a_13 <dbl>, bisec_b_13 <dbl>, bisec_fa_13 <dbl>,
## # bisec_b_14 <dbl>, bisec_fa_14 <dbl>, bisec_fb_14 <dbl>, bisec_a_15 <dbl>, bisec_b_15 <dbl>, bisec_fa_15 <dbl>,
## # bisec_b_16 <dbl>, bisec_fa_16 <dbl>, bisec_fb_16 <dbl>, bisec_a_17 <dbl>, bisec_b_17 <dbl>, bisec_fa_17 <dbl>,
## # bisec_b_18 <dbl>, bisec_fa_18 <dbl>, bisec_fb_18 <dbl>, bisec_a_19 <dbl>, bisec_b_19 <dbl>, bisec_fa_19 <dbl>,
## # bisec_b_20 <dbl>, bisec_fa_20 <dbl>, bisec_fb_20 <dbl>, bisec_a_21 <dbl>, bisec_b_21 <dbl>, bisec_fa_21 <dbl>,
## # bisec_b_22 <dbl>, bisec_fa_22 <dbl>, bisec_fb_22 <dbl>, bisec_a_23 <dbl>, bisec_b_23 <dbl>, bisec_fa_23 <dbl>,
## # bisec_b_24 <dbl>, bisec_fa_24 <dbl>, bisec_fb_24 <dbl>
```

```
str(tb_states_choices_bisec)
```

```
## Classes 'rowwise_df', 'tbl_df', 'tbl' and 'data.frame':  4 obs. of  106 variables:
## $ INDI_ID      : int  1 2 3 4
## $ fl_A         : num  -2 -0.667 0.667 2
## $ fl_alpha     : num  0.1 0.367 0.633 0.9
## $ bisec_a_0    : num  0 0 0 0
## $ bisec_b_0    : num  100 100 100 100
## $ bisec_fa_0   : num  100 100 100 100
## $ bisec_fb_0   : num  -13465 -118529 -2087311 -45152754
## $ p           : num  0.4803 0.2201 0.0821 0.0364
## $ f_p         : num  -0.000163 -0.0007 -0.002141 0.0108
```

```

## $ f_p_t_f_a : num -1.52e-07 -1.59e-06 -1.58e-05 3.86e-04
## $ bisec_a_1 : num 0 0 0 0
## $ bisec_b_1 : num 50 50 50 50
## $ bisec_fa_1 : num 100 100 100 100
## $ bisec_fb_1 : num -6939 -51673 -763570 -13865343
## $ bisec_a_2 : num 0 0 0 0
## $ bisec_b_2 : num 25 25 25 25
## $ bisec_fa_2 : num 100 100 100 100
## $ bisec_fb_2 : num -3569 -22608 -280536 -4273227
## $ bisec_a_3 : num 0 0 0 0
## $ bisec_b_3 : num 12.5 12.5 12.5 12.5
## $ bisec_fa_3 : num 100 100 100 100
## $ bisec_fb_3 : num -1822 -9917 -103579 -1322847
## $ bisec_a_4 : num 0 0 0 0
## $ bisec_b_4 : num 6.25 6.25 6.25 6.25
## $ bisec_fa_4 : num 100 100 100 100
## $ bisec_fb_4 : num -913 -4348 -38447 -411717
## $ bisec_a_5 : num 0 0 0 0
## $ bisec_b_5 : num 3.12 3.12 3.12 3.12
## $ bisec_fa_5 : num 100 100 100 100
## $ bisec_fb_5 : num -437 -1890 -14339 -128960
## $ bisec_a_6 : num 0 0 0 0
## $ bisec_b_6 : num 1.56 1.56 1.56 1.56
## $ bisec_fa_6 : num 100 100 100 100
## $ bisec_fb_6 : num -186 -798 -5357 -40682
## $ bisec_a_7 : num 0 0 0 0
## $ bisec_b_7 : num 0.781 0.781 0.781 0.781
## $ bisec_fa_7 : num 100 100 100 100
## $ bisec_fb_7 : num -53.9 -309.1 -1983.8 -12916.9
## $ bisec_a_8 : num 0.391 0 0 0
## $ bisec_b_8 : num 0.781 0.391 0.391 0.391
## $ bisec_fa_8 : num 16.6 100 100 100
## $ bisec_fb_8 : num -53.9 -88.4 -705.4 -4105.8
## $ bisec_a_9 : num 0.391 0.195 0 0
## $ bisec_b_9 : num 0.586 0.391 0.195 0.195
## $ bisec_fa_9 : num 16.6 12.2 100 100
## $ bisec_fb_9 : num -19.2 -88.4 -215.6 -1279
## $ bisec_a_10 : num 0.391 0.195 0 0
## $ bisec_b_10 : num 0.4883 0.293 0.0977 0.0977
## $ bisec_fa_10 : num 16.6 12.2 100 100
## $ bisec_fb_10 : num -1.47 -36.98 -25.66 -360.05
## $ bisec_a_11 : num 0.4395 0.1953 0.0488 0
## $ bisec_b_11 : num 0.4883 0.2441 0.0977 0.0488
## $ bisec_fa_11 : num 7.53 12.25 49.09 100
## $ bisec_fb_11 : num -1.47 -12.06 -25.66 -56.62
## $ bisec_a_12 : num 0.4639 0.2197 0.0732 0.0244
## $ bisec_b_12 : num 0.4883 0.2441 0.0977 0.0488
## $ bisec_fa_12 : num 3.02 0.174 13.828 45.431
## $ bisec_fb_12 : num -1.47 -12.06 -25.66 -56.62
## $ bisec_a_13 : num 0.4761 0.2197 0.0732 0.0244
## $ bisec_b_13 : num 0.4883 0.2319 0.0854 0.0366
## $ bisec_fa_13 : num 0.772 0.174 13.828 45.431
## $ bisec_fb_13 : num -1.47 -5.925 -5.435 -0.813
## $ bisec_a_14 : num 0.4761 0.2197 0.0793 0.0305

```

```
## $ bisec_b_14 : num 0.4822 0.2258 0.0854 0.0366
## $ bisec_fa_14: num 0.772 0.174 4.322 23.583
## $ bisec_fb_14: num -0.349 -2.87 -5.435 -0.813
## $ bisec_a_15 : num 0.4791 0.2197 0.0793 0.0336
## $ bisec_b_15 : num 0.4822 0.2228 0.0824 0.0366
## $ bisec_fa_15: num 0.211 0.174 4.322 11.692
## $ bisec_fb_15: num -0.349 -1.347 -0.526 -0.813
## $ bisec_a_16 : num 0.4791 0.2197 0.0809 0.0351
## $ bisec_b_16 : num 0.4807 0.2213 0.0824 0.0366
## $ bisec_fa_16: num 0.211 0.174 1.906 5.515
## $ bisec_fb_16: num -0.0691 -0.5858 -0.5258 -0.8133
## $ bisec_a_17 : num 0.4799 0.2197 0.0816 0.0359
## $ bisec_b_17 : num 0.4807 0.2205 0.0824 0.0366
## $ bisec_fa_17: num 0.071 0.174 0.692 2.369
## $ bisec_fb_17: num -0.0691 -0.2056 -0.5258 -0.8133
## $ bisec_a_18 : num 0.4803 0.2197 0.082 0.0362
## $ bisec_b_18 : num 0.4807 0.2201 0.0824 0.0366
## $ bisec_fa_18: num 0.000932 0.174451 0.083486 0.782723
## $ bisec_fb_18: num -0.0691 -0.0155 -0.5258 -0.8133
## $ bisec_a_19 : num 0.4803 0.2199 0.082 0.0362
## $ bisec_b_19 : num 0.4805 0.2201 0.0822 0.0364
## $ bisec_fa_19: num 0.000932 0.079458 0.083486 0.782723
## $ bisec_fb_19: num -0.0341 -0.0155 -0.2211 -0.0141
## $ bisec_a_20 : num 0.4803 0.22 0.082 0.0363
## $ bisec_b_20 : num 0.4804 0.2201 0.0821 0.0364
## $ bisec_fa_20: num 0.000932 0.031958 0.083486 0.384585
## $ bisec_fb_20: num -0.0166 -0.0155 -0.0688 -0.0141
## $ bisec_a_21 : num 0.4803 0.2201 0.0821 0.0364
## $ bisec_b_21 : num 0.4803 0.2201 0.0821 0.0364
## $ bisec_fa_21: num 0.000932 0.008207 0.007374 0.185297
## $ bisec_fb_21: num -0.00783 -0.01555 -0.06875 -0.01414
## $ bisec_a_22 : num 0.4803 0.2201 0.0821 0.0364
## $ bisec_b_22 : num 0.4803 0.2201 0.0821 0.0364
## $ bisec_fa_22: num 0.000932 0.008207 0.007374 0.085598
## $ bisec_fb_22: num -0.00345 -0.00367 -0.03069 -0.01414
## $ bisec_a_23 : num 0.4803 0.2201 0.0821 0.0364
## [list output truncated]
```

Table Two—Very Wide to Very Long We want to treat the iteration count information that is the suffix of variable names as a variable by itself. Additionally, we want to treat the a,b,fa,fb as a variable. Structuring the data very long like this allows for easy graphing and other types of analysis. Rather than dealing with many many variables, we have only 3 core variables that store bisection iteration information.

Here we use the very nice *pivot_longer* function. Note that to achieve this, we put a common prefix in front of the variables we wanted to convert to long. This is helpful, because we can easily identify which variables need to be reshaped.

```
# New variables
svr_bisect_iter <- 'biseciter'
svr_abfafb_long_name <- 'varname'
svr_number_col <- 'value'
svr_id_bisect_iter <- paste0(svr_id_var, '_bisect_ier')

# Pivot wide to very long
tb_states_choices_bisec_long <- tb_states_choices_bisec %>%
```

```

pivot_longer(
  cols = starts_with(st_bisec_prefix),
  names_to = c(svr_abfafb_long_name, svr_bisect_iter),
  names_pattern = paste0(st_bisec_prefix, "(.*)_(.*)"),
  values_to = svr_number_col
)

# Print
summary(tb_states_choices_bisec_long)

```

```

##      INDI_ID      fl_A      fl_alpha      p      f_p      f_p_t_f_a
## Min.   :1.00   Min.   :-2   Min.   :0.1   Min.   :0.03642   Min.   :-0.0021411   Min.   :-1.579e-05
## 1st Qu.:1.75   1st Qu.: -1   1st Qu.:0.3   1st Qu.:0.07066   1st Qu.: -0.0010605   1st Qu.: -5.139e-06
## Median :2.50   Median : 0   Median :0.5   Median :0.15107   Median : -0.0004316   Median : -8.702e-07
## Mean   :2.50   Mean   : 0   Mean   :0.5   Mean   :0.20471   Mean   : 0.0019490   Mean   : 9.211e-05
## 3rd Qu.:3.25   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.:0.28513   3rd Qu.: 0.0025780   3rd Qu.: 9.637e-05
## Max.   :4.00   Max.   : 2   Max.   :0.9   Max.   :0.48028   Max.   : 0.0108004   Max.   : 3.860e-04
##      value
## Min.   :-45152754
## 1st Qu.:      0
## Median :      0
## Mean   : -171858
## 3rd Qu.:      0
## Max.   :      100

```

```

head(tb_states_choices_bisec_long %>% select(-one_of('p', 'f_p', 'f_p_t_f_a')), 30)

```

```

## # A tibble: 30 x 6
##      INDI_ID fl_A fl_alpha varname biseciter value
##      <int> <dbl>   <dbl> <chr>   <chr>   <dbl>
## 1         1    -2     0.1 a       0         0
## 2         1    -2     0.1 b       0        100
## 3         1    -2     0.1 fa      0        100
## 4         1    -2     0.1 fb      0    -13465.
## 5         1    -2     0.1 a       1         0
## 6         1    -2     0.1 b       1         50
## 7         1    -2     0.1 fa      1        100
## 8         1    -2     0.1 fb      1    -6939.
## 9         1    -2     0.1 a       2         0
## 10        1    -2     0.1 b       2         25
## # ... with 20 more rows

```

```

tail(tb_states_choices_bisec_long %>% select(-one_of('p', 'f_p', 'f_p_t_f_a')), 30)

```

```

## # A tibble: 30 x 6
##      INDI_ID fl_A fl_alpha varname biseciter value
##      <int> <dbl>   <dbl> <chr>   <chr>   <dbl>
## 1         4     2     0.9 fa      17      2.37
## 2         4     2     0.9 fb      17     -0.813
## 3         4     2     0.9 a       18      0.0362
## 4         4     2     0.9 b       18      0.0366
## 5         4     2     0.9 fa      18      0.783
## 6         4     2     0.9 fb      18     -0.813
## 7         4     2     0.9 a       19      0.0362
## 8         4     2     0.9 b       19      0.0364

```

```
## 9      4      2      0.9 fa      19      0.783
## 10     4      2      0.9 fb      19     -0.0141
## # ... with 20 more rows
```

Table Two—Very Very Long to Wider Again But the previous results are too long, with the a, b, fa, and fb all in one column as different categories, they are really not different categories, they are in fact different types of variables. So we want to spread those four categories of this variable into four columns, each one representing the a, b, fa, and fb values. The rows would then be uniquely identified by the iteration counter and individual ID.

```
# Pivot wide to very long to a little wide
tb_states_choices_bisec_wider <- tb_states_choices_bisec_long %>%
  pivot_wider(
    names_from = !!sym(svr_abfafb_long_name),
    values_from = svr_number_col
  )

# Print
summary(tb_states_choices_bisec_wider)
```

```
##      INDI_ID      fl_A      fl_alpha      p      f_p      f_p_t_f_a
## Min.      :1.00    Min.      :-2    Min.      :0.1    Min.      :0.03642    Min.      :-0.0021411    Min.      :-1.579e-05
## 1st Qu.:1.75    1st Qu.: -1    1st Qu.:0.3    1st Qu.:0.07066    1st Qu.: -0.0010605    1st Qu.: -5.139e-06
## Median :2.50    Median : 0    Median :0.5    Median :0.15107    Median : -0.0004316    Median : -8.702e-07
## Mean   :2.50    Mean   : 0    Mean   :0.5    Mean   :0.20471    Mean   : 0.0019490    Mean   : 9.211e-05
## 3rd Qu.:3.25    3rd Qu.: 1    3rd Qu.:0.7    3rd Qu.:0.28513    3rd Qu.: 0.0025780    3rd Qu.: 9.637e-05
## Max.   :4.00    Max.   : 2    Max.   :0.9    Max.   :0.48028    Max.   : 0.0108004    Max.   : 3.860e-04
##      b      fa      fb
## Min.      : 0.03643    Min.      : 0.00093    Min.      : -45152754
## 1st Qu.: 0.08545    1st Qu.: 0.17445    1st Qu.: -1914
## Median : 0.39062    Median : 12.24966    Median : -6
## Mean   : 8.12801    Mean   : 43.25156    Mean   : -687482
## 3rd Qu.: 1.56250    3rd Qu.:100.00000    3rd Qu.: 0
## Max.   :100.00000    Max.   :100.00000    Max.   : 0
```

```
print(tb_states_choices_bisec_wider %>% select(-one_of('p', 'f_p', 'f_p_t_f_a')))
```

```
## # A tibble: 100 x 8
##      INDI_ID fl_A fl_alpha biseciter      a      b      fa      fb
##      <int> <dbl>      <dbl> <chr>      <dbl>      <dbl> <dbl>      <dbl>
## 1      1      -2      0.1 0          0      100      100     -13465.
## 2      1      -2      0.1 1          0       50      100     -6939.
## 3      1      -2      0.1 2          0       25      100     -3569.
## 4      1      -2      0.1 3          0      12.5     100     -1822.
## 5      1      -2      0.1 4          0       6.25     100      -913.
## 6      1      -2      0.1 5          0       3.12     100      -437.
## 7      1      -2      0.1 6          0       1.56     100      -186.
## 8      1      -2      0.1 7          0       0.781     100       -53.9
## 9      1      -2      0.1 8          0.391     0.781     16.6       -53.9
## 10     1      -2      0.1 9          0.391     0.586     16.6      -19.2
## # ... with 90 more rows
```

```
print(tb_states_choices_bisec_wider %>% select(-one_of('p', 'f_p', 'f_p_t_f_a')))
```

```
## # A tibble: 100 x 8
##      INDI_ID fl_A fl_alpha biseciter      a      b      fa      fb
```



```
##      <int> <dbl>      <dbl> <chr>      <dbl>      <dbl> <dbl>      <dbl>
## 1      1      -2      0.1 0          0      100      100     -13465.
## 2      1      -2      0.1 1          0       50      100     -6939.
## 3      1      -2      0.1 2          0       25      100     -3569.
## 4      1      -2      0.1 3          0      12.5     100     -1822.
## 5      1      -2      0.1 4          0       6.25     100      -913.
## 6      1      -2      0.1 5          0       3.12     100      -437.
## 7      1      -2      0.1 6          0       1.56     100      -186.
## 8      1      -2      0.1 7          0       0.781    100       -53.9
## 9      1      -2      0.1 8          0.391    0.781    16.6      -53.9
## 10     1      -2      0.1 9          0.391    0.586    16.6      -19.2
## # ... with 90 more rows
```

Graph Bisection Iteration Results Actually we want to graph based on the long results, not the wider. Wider easier to view in table.

```
# Graph results
lineplot <- tb_states_choices_bisec_long %>%
  mutate(!sym(svr_bisect_iter) := as.numeric(!sym(svr_bisect_iter))) %>%
  filter(!sym(svr_abfafb_long_name) %in% c('a', 'b')) %>%
  ggplot(aes(x=!sym(svr_bisect_iter), y=!sym(svr_number_col),
    colour=!sym(svr_abfafb_long_name),
    linetype=!sym(svr_abfafb_long_name),
    shape=!sym(svr_abfafb_long_name))) +
  facet_wrap( ~ INDI_ID) +
  geom_line() +
  geom_point() +
  labs(title = 'Bisection Iteration over individuals Until Convergence',
    x = 'Bisection Iteration',
    y = 'a (left side point) and b (right side point) values',
    caption = 'DPLYR concurrent bisection nonlinear multiple individuals') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(lineplot)
```

Bisection Iteration over individuals Until Convergence

