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

Fan Wang

Contents

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 resuable 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. iteratre until convergence.

Generate New columns of a and b as we iteratre, 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)
```

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 interations to allow at most
fl_tol <- 10^-2
it_tol <- 100

# fl_p_dist2zr = distance to zero to initalize
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)
# str(tb_states_choices_bisec)
```

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)
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 uniquly 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)
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
```

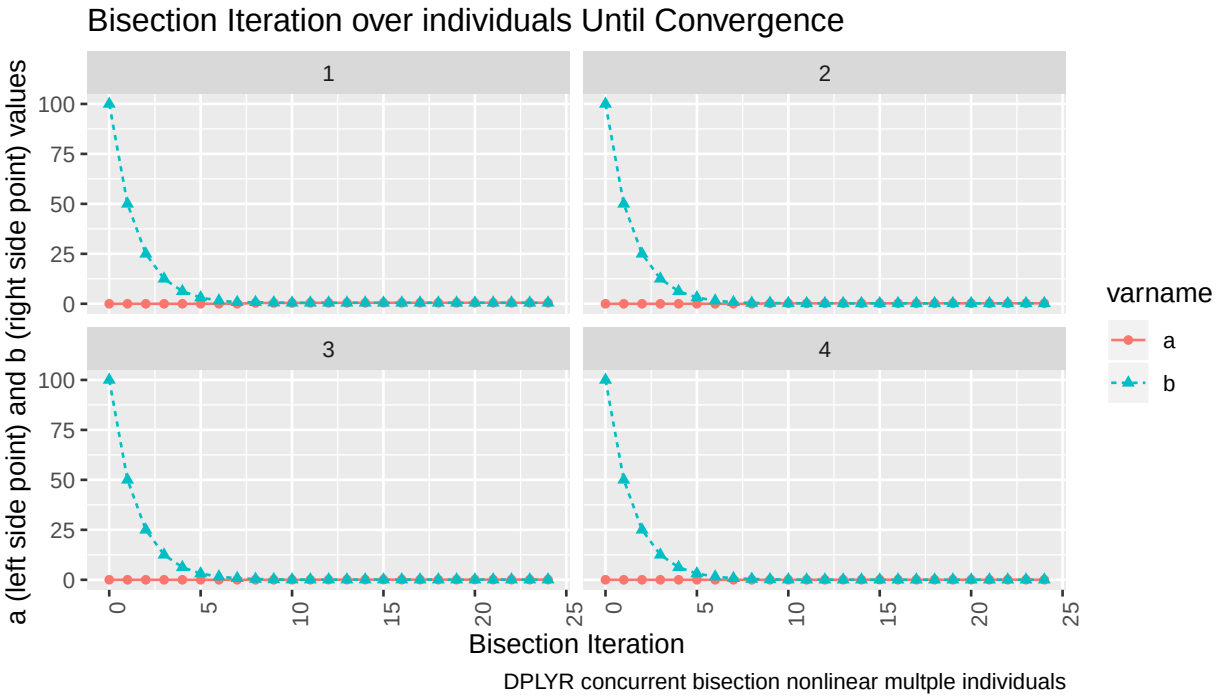
```
print(tb_states_choices_bisec_wider %>% select(-one_of('p','f_p','f_p_t_f_a')))
```

```
## # 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 multple individuals') +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(lineplot)
```


Bisection Iteration over individuals Until Convergence
1
2
3
4
a (left side point) and b (right side point) values
100
75
50
25
0
0
5
10
15
20
25
Bisection Iteration
varname
a
b
DPLYR concurrent bisection nonlinear multple individuals