

# Evaluate Functions over Rows of Meshes Matrices in R

Fan Wang

2020-04-19

## Contents

Evaluate Choices Across States . . . . .	1
--	---

### Evaluate Choices Across States

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

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.

We want evaluate linear function  $0 = f(z_{ij}, x_i, y_i, \mathbf{X}, \mathbf{Y}, c, d)$ . There are  $i$  functions that have  $i$  specific  $x$  and  $y$ . For each  $i$  function, we evaluate along a grid of feasible values for  $z$ , over  $j \in J$  grid points, potentially looking for the  $j$  that is closest to the root.  $\mathbf{X}$  and  $\mathbf{Y}$  are arrays common across the  $i$  equations, and  $c$  and  $d$  are constants.

The evaluation strategy is the following, given min and max for  $z$  that are specific for each  $j$ , and given common number of grid points, generate a matrix of  $z_{ij}$ . Suppose there the number of  $i$  is  $I$ , and the number of grid points for  $j$  is  $J$ .

1. Generate a  $J \cdot I$  by 3 matrix where the columns are  $z, x, y$  as tibble
2. Follow [this](#) Mutate to evaluate the  $f(\cdot)$  function.
3. Add two categorical columns for grid levels and with  $i, i$  and  $j$  index. Plot Mutate output evaluated column categorized by  $i$  as color and  $j$  as x-axis.

**Set up Input Arrays** There is a function that takes  $M = Q + P$  inputs, we want to evaluate this function  $N$  times. Each time, there are  $M$  inputs, where all but  $Q$  of the  $M$  inputs, meaning  $P$  of the  $M$  inputs, are the same. In particular,  $P = Q * N$ .

$$M = Q + P = Q + Q * N$$

Now we need to expand this by the number of choice grid. Each row, representing one equation, is expanded by the number of choice grids. We are graphically searching, or rather brute force searching, which means if we have 100 individuals, we want to plot out the nonlinear equation for each of these lines, and show graphically where each line crosses zero. We achieve this, by evaluating the equation for each of the 100 individuals along a grid of feasible choices.

In this problem here, the feasible choices are shared across individuals.

```
# Parameters
fl_rho = 0.20
svr_id_var = 'INDI_ID'

# it_child_count = N, the number of children
```

```

it_N_child_cnt = 4
# it_heter_param = Q, number of parameters that are heterogeneous across children
it_Q_hetpa_cnt = 2

# P fixed parameters, nN is N dimensional, nP is P dimensional
ar_nN_A = seq(-2, 2, length.out = it_N_child_cnt)
ar_nN_alpha = seq(0.1, 0.9, length.out = it_N_child_cnt)
ar_nP_A_alpha = c(ar_nN_A, ar_nN_alpha)

# N by Q varying parameters
mt_nN_by_nQ_A_alpha = cbind(ar_nN_A, ar_nN_alpha)

# Choice Grid for nutritional feasible choices for each
fl_N_agg = 100
fl_N_min = 0
it_N_choice_cnt_ttest = 3
it_N_choice_cnt_dense = 100
ar_N_choices_ttest = seq(fl_N_min, fl_N_agg, length.out = it_N_choice_cnt_ttest)
ar_N_choices_dense = seq(fl_N_min, fl_N_agg, length.out = it_N_choice_cnt_dense)

# Mesh Expand
tb_states_choices <- as_tibble(mt_nN_by_nQ_A_alpha) %>% rowid_to_column(var=svr_id_var)
tb_states_choices_ttest <- tb_states_choices %>% expand_grid(choices = ar_N_choices_ttest)
tb_states_choices_dense <- tb_states_choices %>% expand_grid(choices = ar_N_choices_dense)

# display
summary(tb_states_choices_dense)

##      INDI_ID      ar_nN_A      ar_nN_alpha      choices
##  Min.   :1.00   Min.   :-2    Min.    :0.1   Min.    : 0
##  1st Qu.:1.75   1st Qu.: -1   1st Qu.:0.3   1st Qu.: 25
##  Median :2.50   Median : 0    Median :0.5   Median : 50
##  Mean   :2.50   Mean    : 0    Mean    :0.5   Mean    : 50
##  3rd Qu.:3.25   3rd Qu.: 1    3rd Qu.:0.7   3rd Qu.: 75
##  Max.   :4.00   Max.    : 2    Max.    :0.9   Max.    :100

kable(tb_states_choices_ttest) %>%
  kable_styling_fc()

```

INDI_ID	ar_nN_A	ar_nN_alpha	choices
1	-2.0000000	0.1000000	0
1	-2.0000000	0.1000000	50
1	-2.0000000	0.1000000	100
2	-0.6666667	0.3666667	0
2	-0.6666667	0.3666667	50
2	-0.6666667	0.3666667	100
3	0.6666667	0.6333333	0
3	0.6666667	0.6333333	50
3	0.6666667	0.6333333	100
4	2.0000000	0.9000000	0
4	2.0000000	0.9000000	50
4	2.0000000	0.9000000	100

**Apply Same Function all Rows, Some Inputs Row-specific, other Shared** There are two types of inputs, row-specific inputs, and inputs that should be applied for each row. The Function just requires all of these inputs, it does not know what is row-specific and what is common for all row. Dplyr recognizes which parameter inputs already existing in the piped dataframe/tibble, given rowwise, those will be row-specific inputs. Additional function parameters that do not exist in dataframe as variable names, but that are pre-defined scalars or arrays will be applied to all rows.

- @param 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: *fl\_A*, *fl\_alpha*, *fl\_N*
- @param scalar and array values that are applied to every rowwise calculation, all rowwise calculations using the same scalars and arrays: *ar\_A*, *ar\_alpha*, *fl\_N\_agg*, *fl\_rho*
- @param string output variable name

The function looks within group, finds min/max etc that are relevant.

```
# Convert Matrix to Tibble
ar_st_col_names = c(svr_id_var, 'fl_A', 'fl_alpha')
tb_states_choices <- tb_states_choices %>% rename_all(~c(ar_st_col_names))
ar_st_col_names = c(svr_id_var, 'fl_A', 'fl_alpha', 'fl_N')
tb_states_choices_ttest <- tb_states_choices_ttest %>% rename_all(~c(ar_st_col_names))
tb_states_choices_dense <- tb_states_choices_dense %>% rename_all(~c(ar_st_col_names))

# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, ar_A, ar_alpha, fl_N_agg, fl_rho){
  # scalar value that are row-specific, in dataframe already: *fl_A*, *fl_alpha*, *fl_N*
  # array and scalars not in dataframe, common all rows: *ar_A*, *ar_alpha*, *fl_N_agg*, *fl_rho*

  # Test Parameters
  # ar_A = ar_nN_A
  # ar_alpha = ar_nN_alpha
  # fl_N = 100
  # fl_rho = -1
  # fl_N_q = 10

  # Apply Function
  ar_p1_s1 = exp((fl_A - ar_A)*fl_rho)
  ar_p1_s2 = (fl_alpha/ar_alpha)
  ar_p1_s3 = (1/(ar_alpha*fl_rho - 1))
  ar_p1 = (ar_p1_s1*ar_p1_s2)^ar_p1_s3
  ar_p2 = fl_N^((fl_alpha*fl_rho-1)/(ar_alpha*fl_rho-1))
  ar_overall = ar_p1*ar_p2
  fl_overall = fl_N_agg - sum(ar_overall)

  return(fl_overall)
}
```

### 3 Points and Denser Dataframes and Define Function

**Evaluate at Three Choice Points and Show Table** In the example below, just show results evaluating over three choice points and show table.

```
# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_states_choices_ttest_eval = tb_states_choices_ttest %>% rowwise() %>%
  mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N,
```

```

ar_nN_A, ar_nN_alpha,
fl_N_agg, fl_rho))

# Show
kable(tb_states_choices_ttest_eval) %>%
  kable_styling_fc()

```

INDI_ID	fl_A	fl_alpha	fl_N	dplyr_eval
1	-2.0000000	0.1000000	0	100.00000
1	-2.0000000	0.1000000	50	-5666.95576
1	-2.0000000	0.1000000	100	-12880.28392
2	-0.6666667	0.3666667	0	100.00000
2	-0.6666667	0.3666667	50	-595.73454
2	-0.6666667	0.3666667	100	-1394.70698
3	0.6666667	0.6333333	0	100.00000
3	0.6666667	0.6333333	50	-106.51058
3	0.6666667	0.6333333	100	-323.94216
4	2.0000000	0.9000000	0	100.00000
4	2.0000000	0.9000000	50	22.55577
4	2.0000000	0.9000000	100	-51.97161

**Evaluate at Many Choice Points and Show Graphically** Same as above, but now we evaluate the function over the individuals at many choice points so that we can graph things out.

```

# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_states_choices_dense_eval = tb_states_choices_dense %>% rowwise() %>%
  mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N,
    ar_nN_A, ar_nN_alpha,
    fl_N_agg, fl_rho))

```

```

# Labeling
st_title <- paste0('Evaluate Non-Linear Functions to Search for Roots')
st_subtitle <- paste0('https://fanwangecon.github.io/',
  'R4Econ/function/mutateef/htmlpdf/fs_func_choice_states.html')
st_caption <- paste0('Evaluating the function, ',
  'https://fanwangecon.github.io/R4Econ/')
st_x_label <- 'x values'
st_y_label <- 'f(x)'

```

```

# Show
dim(tb_states_choices_dense_eval)

```

```
## [1] 400 5
```

```
summary(tb_states_choices_dense_eval)
```

```

##      INDI_ID      fl_A      fl_alpha      fl_N      dplyr_eval
##  Min.   :1.00  Min.   :-2    Min.   :0.1    Min.   : 0    Min.   :-12880.28
##  1st Qu.:1.75  1st Qu.: -1    1st Qu.:0.3    1st Qu.: 25    1st Qu.: -1167.29
##  Median :2.50  Median : 0    Median :0.5    Median : 50    Median : -202.42
##  Mean   :2.50  Mean   : 0    Mean   :0.5    Mean   : 50    Mean   : -1645.65
##  3rd Qu.:3.25  3rd Qu.: 1    3rd Qu.:0.7    3rd Qu.: 75    3rd Qu.:    0.96
##  Max.   :4.00  Max.   : 2    Max.   :0.9    Max.   :100    Max.   :   100.00

```

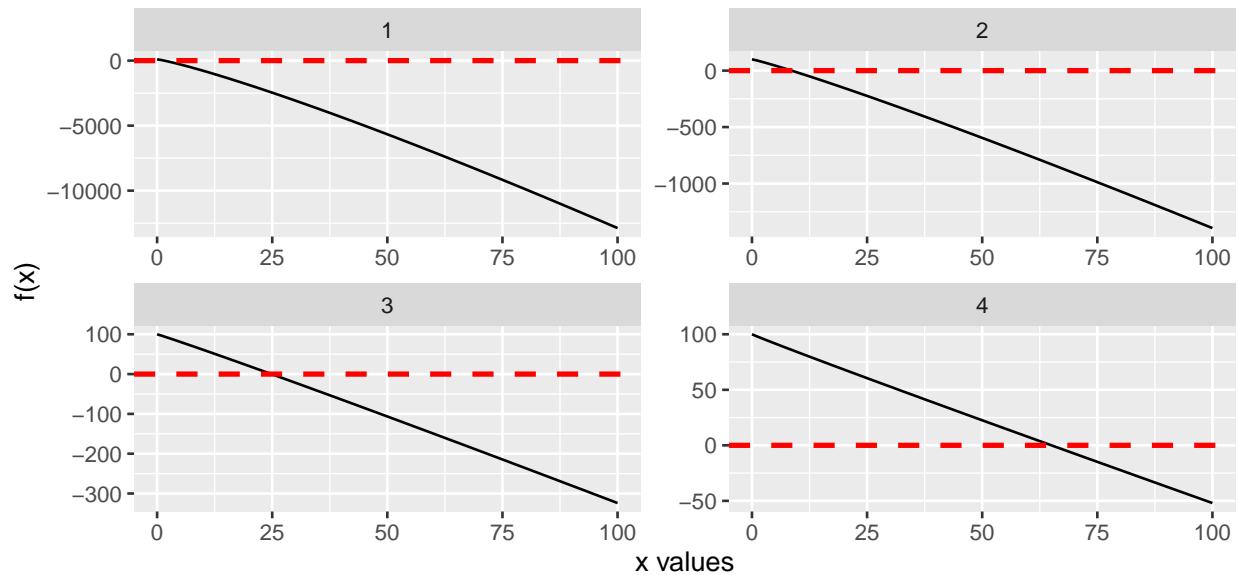
```

lineplot <- tb_states_choices_dense_eval %>%
  ggplot(aes(x=fl_N, y=dplyr_eval)) +
  geom_line() +
  facet_wrap( . ~ INDI_ID, scales = "free") +
  geom_hline(yintercept=0, linetype="dashed",
    color = "red", size=1) +
  labs(title = st_title,
    subtitle = st_subtitle,
    x = st_x_label,
    y = st_y_label,
    caption = st_caption)
print(lineplot)

```

## Evaluate Non-Linear Functions to Search for Roots

[https://fanwangecon.github.io/R4Econ/function/mutatef/htmlpdf/fs\\_func\\_choice\\_states.html](https://fanwangecon.github.io/R4Econ/function/mutatef/htmlpdf/fs_func_choice_states.html)



Evaluating the function, <https://fanwangecon.github.io/R4Econ/>