Evaluate a Many One Unknown Nonlinear Equations Jointly, Brute Force Find the Roots of All Nonlinear Equations

Go back to fan's R4Econ Repository or Intro Stats with R Repository.

Issue and Goal

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 wich i, i and j index. Plot Mutate output evaluated column categorized by i as color and j as x-axis.

Set Up

```r
rm(list = ls(all.names = TRUE))
options(knitr.duplicate.label = 'allow')
```

```r
library(tidyverse)
library(tidyr)
library(knitr)
library(kableExtra)
# file name
st_file_name = 'fs_func_graph_eval'
# Generate R File
purl(paste0(st_file_name, ".Rmd"), output=paste0(st_file_name, ".R"), documentation = 2)
# Generate PDF and HTML
# rmarkdown::render("C:/Users/fan/R4Econ/support/function/fs_funceval.Rmd", "pdf_document")
# rmarkdown::render("C:/Users/fan/R4Econ/support/function/fs_funceval.Rmd", "html_document")
```

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 = 5
# 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    Min.   :-2   Min.   :0.1   Min.   :  0
##  1st Qu.:2    1st Qu.:-1   1st Qu.:0.3   1st Qu.: 25
##  Median :3    Median : 0   Median :0.5   Median : 50
##  Mean   :3    Mean   : 0   Mean   :0.5   Mean   : 50
##  3rd Qu.:4    3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.: 75
##  Max.   :5    Max.   : 2   Max.   :0.9   Max.   :100
```

```
kable(tb_states_choices_ttest) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

INDI_ID	ar_nN_A	ar_nN_alpha	choices
1	-2	0.1	0
1	-2	0.1	50
1	-2	0.1	100
2	-1	0.3	0
2	-1	0.3	50
2	-1	0.3	100
3	0	0.5	0
3	0	0.5	50
3	0	0.5	100
4	1	0.7	0
4	1	0.7	50
4	1	0.7	100
5	2	0.9	0
5	2	0.9	50
5	2	0.9	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.

3 Points and Denser Dataframs and Define Function

```r
# Convert Matrix to Tibble
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)
}
```

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(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

INDI_ID	fl_A	fl_alpha	fl_N	dplyr_eval
1	-2	0.1	0	100.000000
1	-2	0.1	50	-6659.833703
1	-2	0.1	100	-15057.608006
2	-1	0.3	0	100.000000
2	-1	0.3	50	-1102.665285
2	-1	0.3	100	-2503.969566
3	0	0.5	0	100.000000
3	0	0.5	50	-330.744393
3	0	0.5	100	-801.333945
4	1	0.7	0	100.000000
4	1	0.7	50	-89.889405
4	1	0.7	100	-284.126293
5	2	0.9	0	100.000000
5	2	0.9	50	7.332908
5	2	0.9	100	-81.252164

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))
```

```
# Show
dim(tb_states_choices_dense_eval)
```

```
## [1] 500   5
```

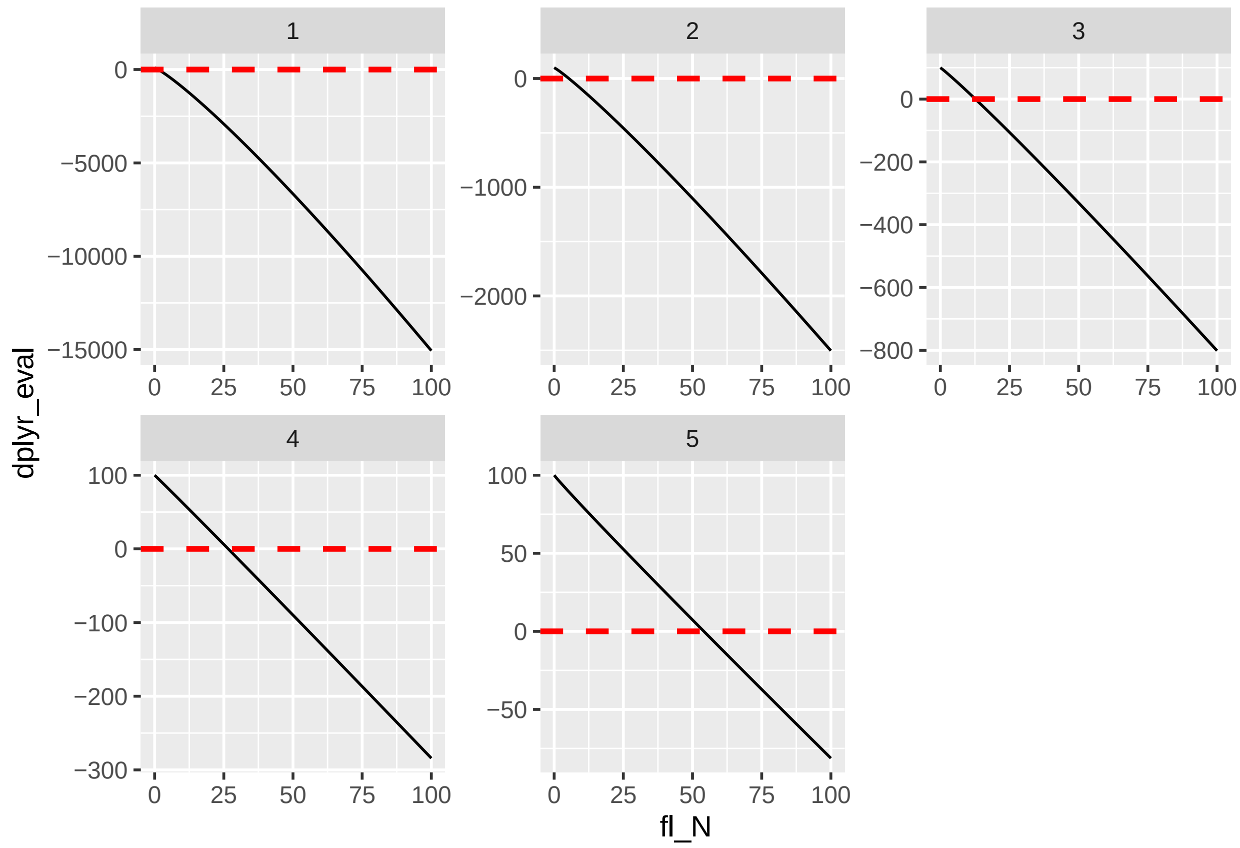
```
summary(tb_states_choices_dense_eval)
```

```
##     INDI_ID       fl_A       fl_alpha        fl_N       dplyr_eval
##  Min.   :1   Min.   :-2   Min.   :0.1   Min.   :  0   Min.   :-15057.61
##  1st Qu.:2   1st Qu.:-1   1st Qu.:0.3   1st Qu.: 25   1st Qu.: -1397.02
##  Median :3   Median : 0   Median :0.5   Median : 50   Median :  -242.82
##  Mean   :3   Mean   : 0   Mean   :0.5   Mean   : 50   Mean   : -1691.91
##  3rd Qu.:4   3rd Qu.: 1   3rd Qu.:0.7   3rd Qu.: 75   3rd Qu.:   -23.75
##  Max.   :5   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 = 'Evaluate Non-Linear Functions to Search for Roots',
             x = 'X values',
             y = 'f(x)',
             caption = 'Evaluating the Function')
```

```
## $x
## [1] "X values"
##
## $y
## [1] "f(x)"
##
## $title
## [1] "Evaluate Non-Linear Functions to Search for Roots"
##
## $caption
## [1] "Evaluating the Function"
##
## attr(,"class")
## [1] "labels"
```

```
print(lineplot)
```



Bisection Solve Optimal Choice for Each Individual

The bisection specific code does not need to do much.

- @param list variables in file for grouping, each group is an individual for whom we want to calculate optimal choice for using bisection.
- @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.
- @param scalar and array values that are applied to every rowwise calculation, all rowwise calculations using the same scalars and arrays.
- @param string output variable name