DPLYR Evaluate Function with N Arrays of Inputs J times

Fan Wang

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

Issue and Goal

We want evaluate nonlinear function f(Q_i, y_i, ar_x, ar_y, c_j, d), where c_j is an element of some array, d is a constant, and ar_x and ar_y are arrays, both fixed. x_i and y_i vary over each row of matrix. We would like to evaluate this nonlinear function concurrently across N individuals. The eventual goal is to find the i specific Q that solves the nonlinear equations.

In Evaluate Nonlinear Function with N arrays of Inputs, we evaluated the function fixing c, the difference here is that we want to evaluate J times the overall matrix of inputs.

The results should be stored in a tibble where each of the $j \in J$ evaluations are stacked together with a variable c that records which c_j value was used for this evaluation.

This is achieved here by using expand_grid.

Set Up

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

```
library(tidyverse)
library(knitr)
library(kableExtra)
# file name
st_file_name = 'fs_funceval'
# 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, $P = Q * N$.

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

```
# 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)
ar_nN_N_choice = seq(1,it_N_child_cnt)/sum(seq(1,it_N_child_cnt))

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

Now generate a vector of ρ, which represents varying planner preference, and mesh it together with the matrix *mt_nN_by_nQ_A_alpha*.

```
# Vector of Planner Preference
ar_rho = c(-0.25, -0.15, 0.15, 0.25)

# N by Q varying parameters but now Mesh with RHO, the J elements we want to vary over
mt_nN_by_nQ_A_alpha_mesh_rho <- as_tibble(mt_nN_by_nQ_A_alpha) %>% expand_grid(rho = ar_rho) %>%
                                  arrange(rho, ar_nN_A, ar_nN_alpha, ar_nN_N_choice)

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

ar_nN_A	ar_nN_alpha	ar_nN_N_choice	rho
-2	0.1	0.0666667	-0.25
-1	0.3	0.1333333	-0.25
0	0.5	0.2000000	-0.25
1	0.7	0.2666667	-0.25
2	0.9	0.3333333	-0.25
-2	0.1	0.0666667	-0.15
-1	0.3	0.1333333	-0.15
0	0.5	0.2000000	-0.15
1	0.7	0.2666667	-0.15
2	0.9	0.3333333	-0.15
-2	0.1	0.0666667	0.15
-1	0.3	0.1333333	0.15
0	0.5	0.2000000	0.15
1	0.7	0.2666667	0.15
2	0.9	0.3333333	0.15
-2	0.1	0.0666667	0.25
-1	0.3	0.1333333	0.25
0	0.5	0.2000000	0.25
1	0.7	0.2666667	0.25
2	0.9	0.3333333	0.25

And the aggregate resources available for allocations, along with observed allocations.

```
# Total Resources available
fl_N_agg = 100
```

Define Function

```
# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, fl_rho, ar_A, ar_alpha, fl_N_agg){

  # 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_N_agg)^((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 Nonlinear Function using dplyr mutate Once

Below, repeat what we did in Evaluate Nonlinear Function with N arrays of Inputs.

```
# Convert Matrix to Tibble
ar_st_col_names = c('fl_A', 'fl_alpha', 'fl_N')
tb_nN_by_nQ_A_alpha <- as_tibble(mt_nN_by_nQ_A_alpha) %>% rename_all(~c(ar_st_col_names))

# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
fl_rho_here = ar_rho[1]
tb_nN_by_nQ_A_alpha = tb_nN_by_nQ_A_alpha %>% rowwise() %>%
                        mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N, fl_rho_here,
                                                               ar_nN_A, ar_nN_alpha,
                                                               fl_N_agg))
# Show
kable(tb_nN_by_nQ_A_alpha) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

fl_A	fl_alpha	fl_N	dplyr_eval
-2	0.1	0.0666667	37.31686
-1	0.3	0.1333333	37.79243
0	0.5	0.2000000	17.11137
1	0.7	0.2666667	-18.21348
2	0.9	0.3333333	-73.95521

Evaluate Nonlinear Function using dplyr multiple times and stack results

Evaluate multiple times and stack results.

```
# Convert Matrix to Tibble
ar_st_col_names = c('fl_A', 'fl_alpha', 'fl_N', 'fl_rho')
```

```
tb_nN_by_nQ_A_alpha_mesh_rho <- as_tibble(mt_nN_by_nQ_A_alpha_mesh_rho) %>% rename_all(~c(ar_st_col_nam

# fl_A, fl_alpha are from columns of tb_nN_by_nQ_A_alpha
tb_nN_by_nQ_A_alpha_mesh_rho = tb_nN_by_nQ_A_alpha_mesh_rho %>% rowwise() %>%
                        mutate(dplyr_eval = ffi_nonlin_dplyrdo(fl_A, fl_alpha, fl_N, fl_rho,
                                                               ar_nN_A, ar_nN_alpha,
                                                               fl_N_agg))
# Show
kable(tb_nN_by_nQ_A_alpha_mesh_rho) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

fl_A	fl_alpha	fl_N	fl_rho	dplyr_eval
-2	0.1	0.0666667	-0.25	37.316863
-1	0.3	0.1333333	-0.25	37.792429
0	0.5	0.2000000	-0.25	17.111367
1	0.7	0.2666667	-0.25	-18.213484
2	0.9	0.3333333	-0.25	-73.955215
-2	0.1	0.0666667	-0.15	13.010884
-1	0.3	0.1333333	-0.15	25.426336
0	0.5	0.2000000	-0.15	14.171990
1	0.7	0.2666667	-0.15	-5.214865
2	0.9	0.3333333	-0.15	-32.498236
-2	0.1	0.0666667	0.15	-329.652680
-1	0.3	0.1333333	0.15	-108.179601
0	0.5	0.2000000	0.15	-38.299405
1	0.7	0.2666667	0.15	3.380053
2	0.9	0.3333333	0.15	31.566650
-2	0.1	0.0666667	0.25	-951.057786
-1	0.3	0.1333333	0.25	-285.694559
0	0.5	0.2000000	0.25	-97.790666
1	0.7	0.2666667	0.25	-6.281001
2	0.9	0.3333333	0.25	42.377578