Evaluate Nonlinear Function with N arrays of Inputs

Go back to fan's 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, d), where c and d are constants, 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.

This is a continuation of R use Apply, Sapply and dplyr Mutate to Evaluate one Function Across Rows of a Matrix

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))
```

```r
# N by Q varying parameters
mt_nN_by_nQ_A_alpha = cbind(ar_nN_A, ar_nN_alpha, ar_nN_N_choice)
# Show
kable(mt_nN_by_nQ_A_alpha) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "responsive"))
```

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

Test non-linear Equation

```r
# Test Parameters
fl_N_agg = 100
fl_rho = -1
fl_N_q = ar_nN_N_choice[4]*fl_N_agg
ar_A_alpha = mt_nN_by_nQ_A_alpha[4,]
# Apply Function
ar_p1_s1 = exp((ar_A_alpha[1] - ar_nN_A)*fl_rho)
ar_p1_s2 = (ar_A_alpha[2]/ar_nN_alpha)
ar_p1_s3 = (1/(ar_nN_alpha*fl_rho - 1))
ar_p1 = (ar_p1_s1*ar_p1_s2)^ar_p1_s3
ar_p2 = fl_N_q^((ar_A_alpha[2]*fl_rho-1)/(ar_nN_alpha*fl_rho-1))
ar_overall = ar_p1*ar_p2
fl_overall = fl_N_agg - sum(ar_overall)
print(fl_overall)
```

```
## [1] -598.2559
```

Test non-linear equation in function

Implement the non-linear problem's evaluation using apply over all N individuals.

```r
# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, ar_A, ar_alpha, fl_N_agg, fl_rho){
  # ar_A_alpha[1] is A
  # ar_A_alpha[2] is alpha

  # # Test Parameters
  # 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)
}

# Parameters
fl_rho = -1

# Evaluate Function
print(ffi_nonlin_dplyrdo(mt_nN_by_nQ_A_alpha[1,1],
                         mt_nN_by_nQ_A_alpha[1,2],
                         mt_nN_by_nQ_A_alpha[1,3]*fl_N_agg,
                         ar_nN_A, ar_nN_alpha, fl_N_agg, fl_rho))
```

```
## [1] 81.86645
```

```
for (i in seq(1,dim(mt_nN_by_nQ_A_alpha)[1])){
  fl_eval = ffi_nonlin_dplyrdo(mt_nN_by_nQ_A_alpha[i,1],
                               mt_nN_by_nQ_A_alpha[i,2],
                               mt_nN_by_nQ_A_alpha[i,3]*fl_N_agg,
                               ar_nN_A, ar_nN_alpha, fl_N_agg, fl_rho)
  print(fl_eval)
}
```

```
## [1] 81.86645
## [1] 54.48885
## [1] -65.5619
## [1] -598.2559
## [1] -3154.072
```

Evaluate Nonlinear Function using dplyr mutate

```
# 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))

# Define Implicit Function
ffi_nonlin_dplyrdo <- function(fl_A, fl_alpha, fl_N, 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_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)
}

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

fl_A	fl_alpha	fl_N	dplyr_eval
-2	0.1	0.0666667	81.86645
-1	0.3	0.1333333	54.48885
0	0.5	0.2000000	-65.56190
1	0.7	0.2666667	-598.25595
2	0.9	0.3333333	-3154.07226