Numerical Optimization but with vectors

Asked 1 month ago Modified 1 month ago Viewed 173 times Part of R Language Collective

1

I am new to solving something numerically, so I ask this to get a starter approach to a problem I have really clear.

So suppose that you have this optimization problem:

$$\max_{c_i,\ell_i} \quad U_i(c_i,\ell_i,\varepsilon) = \alpha_i \cdot \ln(c_i - \gamma_c) + (1-\alpha_i) \cdot \ln(\ell_i - \gamma_\ell), \quad \alpha_i = \bar{\alpha} + \varepsilon_i$$
$$\text{s.t. } c_i = (1-\tau)w_i h_i + I_i$$
$$T_i = h_i + \ell_i$$
$$\ell_i > 0, \quad h_i > 0$$

Where you know the values of `\gamma_c` , `\gamma_\ell` , `\tau` , and `\Bar{\alpha}`

I solved it by hand using Lagrange Multipliers and got a closed-form solution. So I have these answers for consumption (`c`), leisure (`\ell`), and labor supply (`h`)

$$\begin{cases} \ell_i^* = & T_i - (\bar{\alpha}+\varepsilon_i)\gamma_h + \frac{(1-\bar{\alpha}-\varepsilon_i)(I_i-\gamma_c)}{(1-\tau)w_i} \\ h_i^* = & (\bar{\alpha}+\varepsilon_i)\gamma_h - \frac{(1-\bar{\alpha}-\varepsilon_i)(I_i-\gamma_c)}{(1-\tau)w_i} \\ c_i^* = & (1-\tau)w_i(\bar{\alpha}+\varepsilon_i)\gamma_h - (1-\bar{\alpha}-\varepsilon_i)(I_i-\gamma_c) + I_i \end{cases}$$

So the thing is that I can compute the optima (`c` , `\ell` , `h`) like this: (I did this in R, but the procedure in Python or Julia can be very similar)

```
library(tidyverse)

w_par = c(4, 0.4)
i_par = c(3, 0.04)
e_par = c(0, 0.01^2)
gamma_l = 8; gamma_c = 50; tau = 0.08; Time = 24; alpha_bar = 0.7;N = 10000
gamma_h = Time - gamma_l
theta_true = c(gamma_h, gamma_c, alpha_bar, sqrt(e_par[2]))

set.seed(1)
df <- data.frame(w = exp(rnorm(n = N, mean = w_par[1], sd = sqrt(w_par[2]))),
                 I = exp(rnorm(n = N, mean = i_par[2], sd = sqrt(i_par[2]))),
                 e = rnorm(n = N, mean = e_par[1], sd = sqrt(e_par[2]))) %>%
  mutate(a = alpha_bar + e,
         h = a*gamma_h - (((1-a)*(I-gamma_c))/((1-tau)*w)),
         L = Time - h,
```

```
C = (1-tau)*w*h+I,
U = a*log(C-gamma_c) + (1-a)*log(L-gamma_l))
```

Ok now take the first part of the dataframe that only contains these 4 variables (`w` , `I` , `e` , `a`), and the parameters.

Is there a way to obtain `h` , `L` , `C` (optima) with an optimizer? What steps should I follow to find the optimal columns? Do the columns obtained with the optimizer have the same values as the ones I got with the closed-form solution?

I don't need a super explicit answer, but something to start figuring out how to do this.

I state this small model because I know there's a closed-form solution. But for work, I have to get the optima for a model that doesn't have a closed-form solution, and all they told me is to solve it numerically (I don't know how to do it, but I am willing to learn)

Thanks in advance !

Edit: There's a typo in my notation instead of T_i is just T

Edit 2: I put the Python tag because I don't mind having it solved in R or python as long I can retrieve U, L, and C

python r math mathematical-optimization numerical-methods Edit tags

Share Edit Follow Close Flag edited Jul 8 at 3:58

asked Jun 13 at 15:59

Jorge Paredes
963 6 12

Have you checked out the `optim` function? – Melissa Key Jun 13 at 16:01

@MelissaKey Yes, but I am not sure how to apply it here because it's constrained – Jorge Paredes Jun 13 at 16:06

Does this help? stats.stackexchange.com/questions/137734/... – Melissa Key Jun 13 at 16:13

I shouldn't make optimization for every row right? – Jorge Paredes Jun 13 at 16:42

1 Perhaps have a look at genSA – MrSmithGoesToWashington Jun 19 at 8:44

1 Answer

Sorted by:
Reset to default

Date modified (newest first)

0

+50

In response to the 2023-07-08 comment asking about getting the other parameters, I submit the following code edits.

Basically, inside `use_apply_and_optim()` save the optim object and then return a vector of U, h, L and C.

```
U_func_3_h_L_C <- function(x, w=NULL, I=NULL, e=NULL, a=NULL){

  h <- x[1]
  L <- 24-x[1]
  C <-(1-tau)*w*h+I

  ## the U to be maximized
  U <- a*log(C-gamma_c) + (1-a)*log(L-gamma_l)
  -U
}

use_apply_and_optim <-
    apply(df,
          1,
          function(DAT){
            optim.obj <-
        optim(c(Time/2, Time/2, 600), fn=U_func_3_h_L_C,

                w = DAT[1],
                I = DAT[2],
                e = DAT[3],
                a = DAT[4],

          method="L-BFGS-B",
          upper=c(Time, Time, Inf),
          lower=c(0,gamma_l, gamma_c ))

            c("U"=-optim.obj$value,
              "h"=optim.obj$par[1],
              "L"= 24-optim.obj$par[1],
              "C"=(1-tau)*DAT[1]*optim.obj$par[1]+DAT[2])})
```

The output is then a 4x10,000 array where the first row is the U of the previous approaches.

```
dim(use_apply_and_optim)
t(use_apply_and_optim[1:4,1:4])

head(cbind(closed_solution = df$U, use_apply_and_optim[1,]))

plot(df$U, use_apply_and_optim[1,])
abline(a=0,b=1,col="blue")


identical(df$U, use_apply_and_optim[1,])

mean((df$U-use_apply_and_optim[1,])^2)
```

Not sure why the down vote, but here's an implementation where x is a vector -- just need to assign the elements as the first few lines in the function.

```
U_func_3 <- function(x, w=NULL, I=NULL, e=NULL, a=NULL){

  h <- x[1]
  L <- 24-x[1]
  C <-(1-tau)*w*h+I


  ## the U to be maximized
  U <- a*log(C-gamma_c) + (1-a)*log(L-gamma_l)
  -U

}



use_apply_and_optim <-
    apply(df,
          1,
          function(DAT){
        -optim(c(Time/2, Time/2, 600), fn=U_func_3,

                 w = DAT[1],
                 I = DAT[2],
                 e = DAT[3],
                 a = DAT[4],


          method="L-BFGS-B",
          upper=c(Time, Time, Inf),
          lower=c(0,gamma_l, gamma_c ))$value}
    )
```

I was able to reduce the problem to just a one-variable optimization problem with a dynamic lower bound. The solution in this case uses `apply()` to go row-wise through the dataset and then `optim()` to take data values to inform the dynamic lower bound. If I read the model and constraints correctly,

- for a given `h` if we know `Time` then `\ell` is determined, and
- for a given `h` if we know `tau` , `w` , and `I` then `c` is determined

Naturally, the quantities inside the natural logs need to be greater than 0, so

- solving `c > gamma_c` makes the dynamic lower bound for `h` and
- solving `l > gamma_c` makes the upper bound for `h` a static `Time - gamma_l` .

```
## define U function
U_func <- function(x, w=NULL, I=NULL, e=NULL, a=NULL){

  h <- x[1]
  L <- Time - x[1]
  C <- (1-tau)*w*h+I

  ## the U to be maximized
  U <- a*log(C-gamma_c) + (1-a)*log(L-gamma_l)
  -U
```

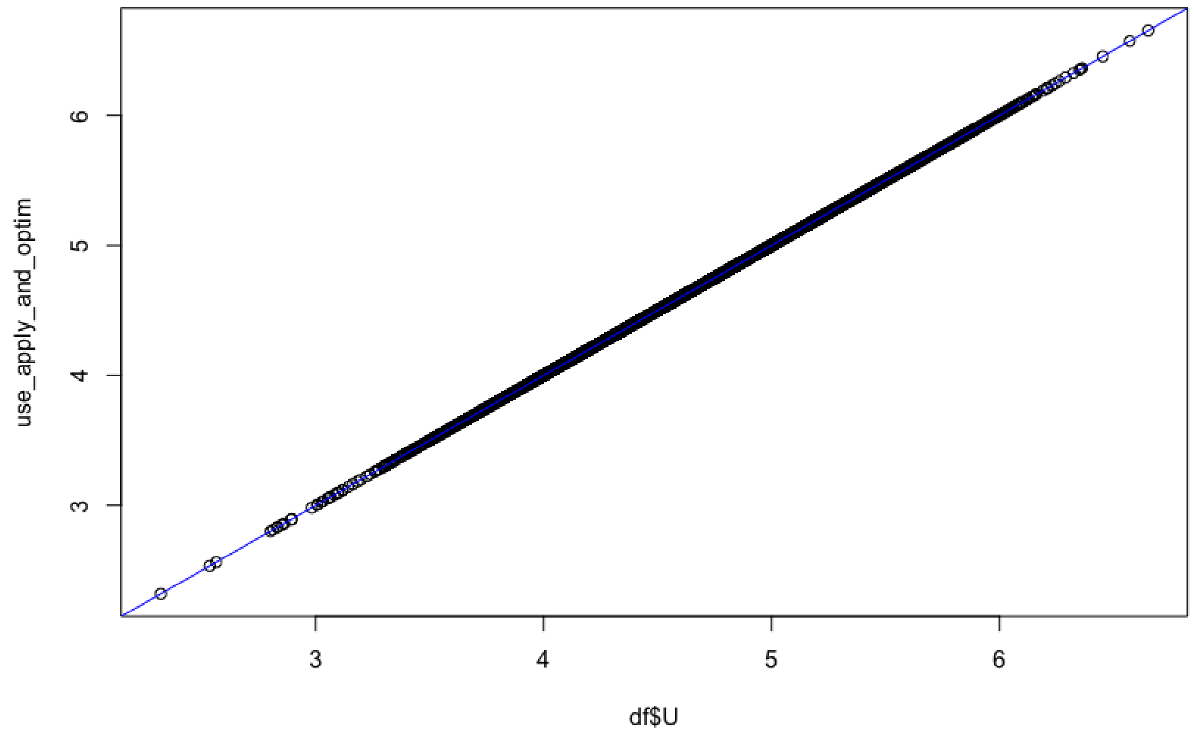
```
}

## store results of optim() via row-wise apply
use_apply_and_optim <-
    apply(df,
          1,
          function(DAT){
        -optim(Time/2, fn=U_func,

              w = DAT[1],
              I = DAT[2],
              e = DAT[3],
              a = DAT[4],

          method="L-BFGS-B",
          upper=c(Time - gamma_l),
          lower=c( (gamma_c - DAT[2])/((1-tau)*DAT[1]) ))$value}
    )


head(cbind(closed_solution = df$U, use_apply_and_optim))
```

Results:

```
> head(cbind(closed_solution = df$U, use_apply_and_optim))
     closed_solution use_apply_and_optim
[1,]        4.541093            4.541093
[2,]        4.940625            4.940625
[3,]        4.396769            4.396769
[4,]        5.476242            5.476242
[5,]        5.050314            5.050314
[6,]        4.419881            4.419881
```

Straight-line:

```
plot(df$U, use_apply_and_optim)
abline(a=0,b=1,col="blue")
```



Not identical, but very low MSE, probably rounding error(?):

```
> identical(df$U, use_apply_and_optim)
[1] FALSE
> mean((df$U-use_apply_and_optim)^2)
[1] 1.127322e-17
```

Share Edit Follow Flag

edited Jul 10 at 13:11

answered Jun 19 at 20:07

swihart
2,628 2 17 41

Let me know if you have more questions -- we can keep working on it together. Best of luck. I've never seen an accepted, bounty-awarded answer with negative votes! Lol. – swihart Jun 27 at 2:50

1 it's running smoothly, but just a question. Is it possible to retrieve L and C that produce the `use_apply_and_optim` vector? Utility is pretty cool, but I need L and C. – Jorge Paredes Jul 8 at 3:50

1 See my edits, let me know if it fills the bill. – swihart Jul 9 at 2:52

1 I edited to remove the negative sign from C. I chose Time/2 fairly arbitrarily -- I figured it is just as good a guess as any other and that they'd each be away from the extremes (0 vs 24). 600 was arbitrary. As for the bounds, the quantities inside the natural logs as written for U need to be greater than 0, so L can't go lower than gamma_l and C can't go below gamma_C. But h is free to vary between 0 and Time. – swihart Jul 10 at 13:16

1 I truly appreciate all the work you've put in this question. Thank you! – Jorge Paredes Jul 10 at 14:19