Optimization Example

Joseph Marlo

12/31/2019

Optimization methodology

This document walks through two examples of optimization. The goal is to find the best age to claim Social Security given that benefits will be reinvested. See `Analysis.Rmd` for a more detailed explanation of the problem.

Note: optimization is not the best method for finding the optimal claim age. Brute force (see `Analysis.Rmd`) is simpler and more comprehensive. There are only nine discrete ages (not counting months) to claim Social Security: ages [62, 70]. Therefore optimization is only performing a one-dimensional search through nine values. However, the purpose of this exercise is to provide a framework for future optimizations.

Setting up the function

We'll be using the `optimize()` and `optim()` functions. The most difficult part is understanding how to set up our function so `optimize()` and `optim()` know what to optimize for. First, since we are looking to maximize the present value of benefits, our function should return a single present value. Second, the first argument to our function should be the input that we are optimizing. In this case, we are trying to find the best `claim.age`.

We can wrap the `calculate_benefits()` function from the `SS_calculator.R` script so it outputs a present value and reorder the arguments so claim age is first. Note that the first argument is renamed `par` so it matches with `optim()` arguments.

```
source("R/SS_calculator.R")
source("Plots/ggplot_themes.R")

opt_function <- function(par, birth.year = 1952, inv.return = 1.05, death.age = 100){
  # function calculates Social Security benefits and returns the present value @ age 62
  # par is the claim age

  # round claim age to nearest integer (fixes issue with continuous optimization)
  par <- round(par)

  # calculate the SS benefits
  benefits.df <- calculate_benefits(birth.year = birth.year, claim.age = par)

  # calculate the present value of those benefits
  present.value <- benefits.df %>%
    filter(Age <= death.age) %>%
    pull(Benefits) %>%
    add_investment_return(., rate = inv.return) %>%
    last(.) %>%
    PV(., rate = inflation.rate, periods = death.age - 62)

  return(present.value)
}
```

```r
# test the function
opt_function(par = 62)
```

```
## [1] 718429.4
```

Continuous method

The simplest optimization approach is to use the one dimensional search function `optimize()`. Since one can only claim Social Security between ages [62, 70] we can set our `interval` so it only searches within these values. We can also set the `tol` or accuracy to a lower value because are we dealing with integers.

Running `optimize()` once returns the best claim age for a given investment return and death age. However, we want to know the best claim age for every combination of investment return and death age. The easiest way to accomplish that is to run `optimize()` for each combination. There may be a better, less computationally intense method for this but for now this will do.

```r
# run the optimization for the default investment return and death age
optimize(f = opt_function, interval = c(62, 70), maximum = TRUE, tol = 0.1)
```

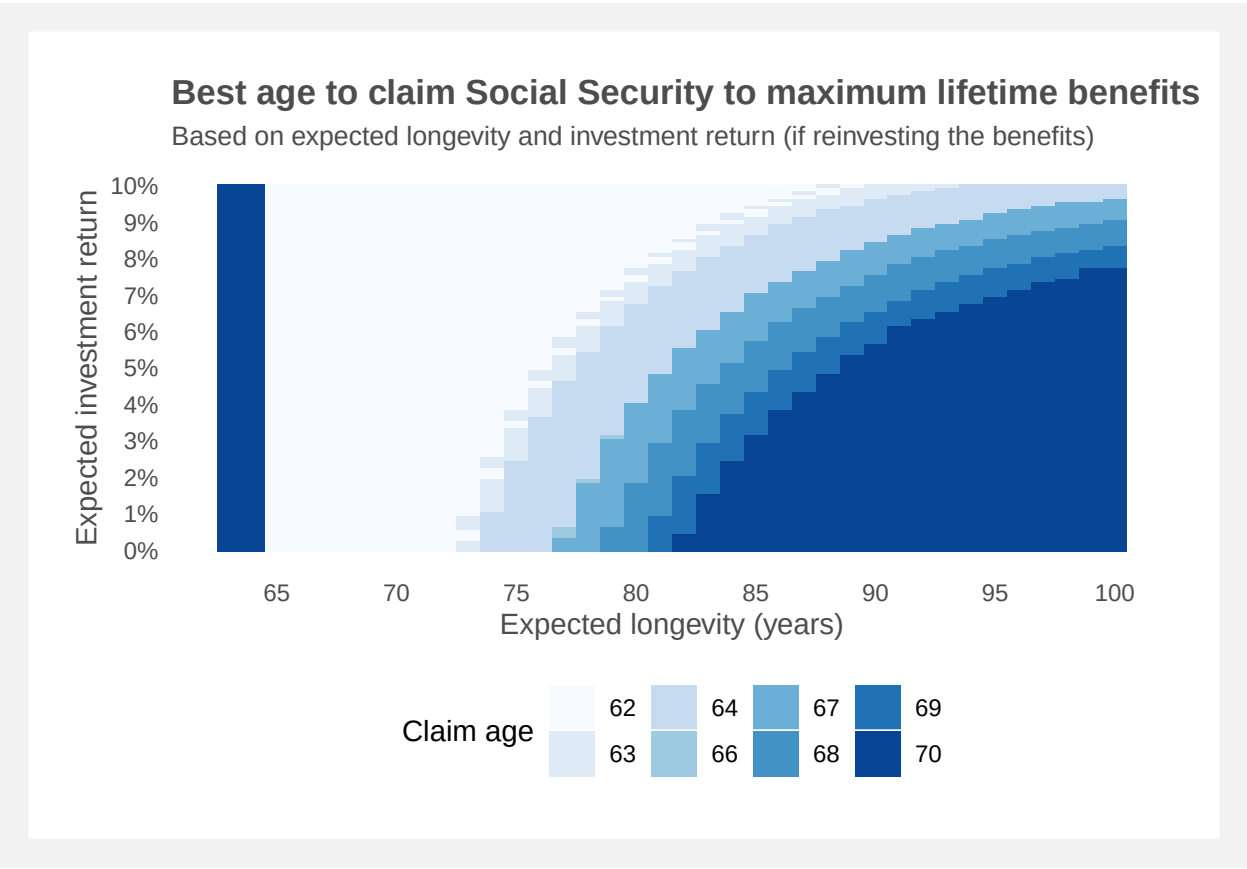
```
## $maximum
## [1] 68.88463
##
## $objective
## [1] 879938.3
```

```r
# create every combination of investment return and death age
params <- expand.grid(Inv.return = seq(1.00, 1.1, by = .001),
                      Death = 63:100)

# optimize for each combination
params$Claim.age <- NA
for (i in 1:nrow(params)) {
  params$Claim.age[[i]] <-
    optimize(
      f = opt_function,
      interval = c(61.5, 70.4),
      maximum = TRUE,
      tol = 0.1,
      inv.return = params$Inv.return[[i]],
      death.age = params$Death[[i]]
    )$maximum %>% round()
}

# tile plot of the results
params %>%
  ggplot(aes(x = Death, y = Inv.return, fill = as.factor(Claim.age))) +
  geom_tile() +
  scale_fill_brewer(name = "Claim age") +
  scale_y_continuous(breaks = seq(1, 1.1, by = 0.01),
                     labels = scales::percent(0:10/100, 1)) +
  scale_x_continuous(breaks = seq(65, 100, 5)) +
  labs(title = "Best age to claim Social Security to maximum lifetime benefits",
       subtitle = "Based on expected longevity and investment return (if reinvesting the benefits)",
       x = "Expected longevity (years)",
       y = "Expected investment return") +
  light.theme +
```

```
  theme(panel.grid.major.y = element_line(color = NA),
        plot.caption = element_text(color = "gray30",
                                    face = "italic",
                                    size = 7),
        legend.position = "bottom")
```



Integer method

If we compare the above plot with the plot from `Analysis.Rmd` then we can see some discrepancies in the optimal claim ages. I believe this is an error due to `calculate_benefits()` requiring an integer input for `claim.age` while `optimize()` is optimizing for continuous variables.

`optim()`, which is usually used for multi-dimensional optimization, can be modified to use integers only. First, we need to create a function that generates the next point to test and specify the underlying method as simulated annealing or "SANN." We specify this generator function to return only integers. Then we can run the optimization similar to the `optimize()` method above with the optional argument `gr` equal to this generator function. Note that birth.year, inv.return, and death.age must be included in this new function even though they are not directly used. This is a requirement by `optim()`.

```
# create function to determine what value to test next (new candidate point)
# returns integers between 62 and 70 (inclusive)
next_fun <- function(x, birth.year, inv.return, death.age) sample(62:70, 1)

# run the optimization for the default investment return and death age
# this is a one-dimensional optimization of the claim.age
optim(
```

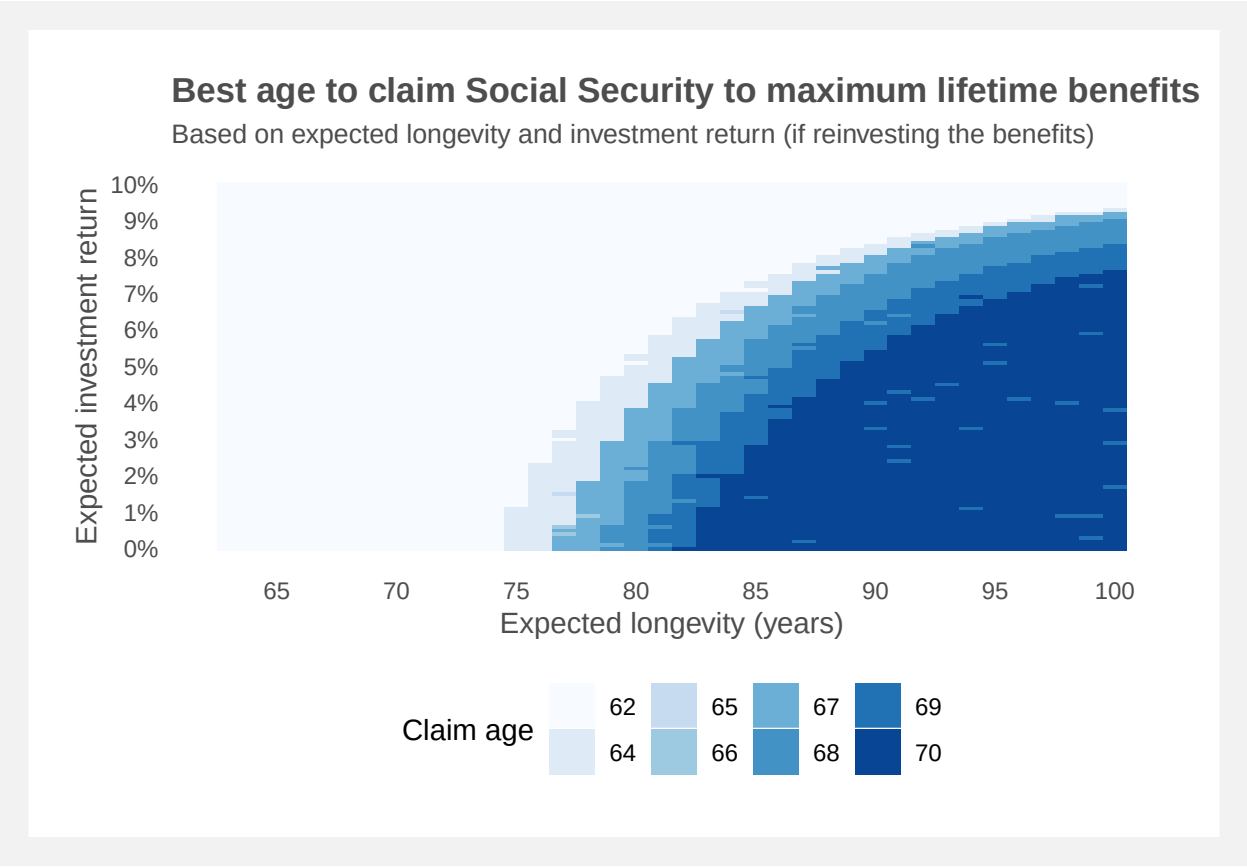
```
  fn = opt_function,
  par = 62,
  birth.year = 1952,
  inv.return = 1.05,
  death.age = 100,
  gr = next_fun,
  method = "SANN",
  control = list(
    maxit = 30, # max number of iterations to work through
    fnscale = -1 # indicates it is a maximization problem
  )
)
```

```
## $par
## [1] 70
##
## $value
## [1] 895215.8
##
## $counts
## function gradient
##       30       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
# optimize for each combination of investment return and death age
params$Int.claim.age <- NA
for (i in 1:nrow(params)) {
  params$Int.claim.age[[i]] <-
    optim(
      fn = opt_function,
      par = 62,
      birth.year = 1952,
      inv.return = params$Inv.return[[i]],
      death.age =  params$Death[[i]],
      gr = next_fun,
      method = "SANN",
      control = list(maxit = 30,
                     fnscale = -1)
    )$par
}

# tile plot of the results
params %>%
  ggplot(aes(x = Death, y = Inv.return, fill = as.factor(Int.claim.age))) +
  geom_tile() +
  scale_fill_brewer(name = "Claim age") +
  scale_y_continuous(breaks = seq(1, 1.1, by = 0.01),
                     labels = scales::percent(0:10/100, 1)) +
  scale_x_continuous(breaks = seq(65, 100, 5)) +
```

```
labs(title = "Best age to claim Social Security to maximum lifetime benefits",
     subtitle = "Based on expected longevity and investment return (if reinvesting the benefits)",
     x = "Expected longevity (years)",
     y = "Expected investment return") +
light.theme +
theme(panel.grid.major.y = element_line(color = NA),
      plot.caption = element_text(color = "gray30",
                                  face = "italic",
                                  size = 7),
      legend.position = "bottom")
```



The results are much closer to the results from `Analysis.Rmd`. However, the process is more complex, the compute time is longer, and it doesn't lend itself to further methodological tweaks such as simulating investment returns. There may be a better method than simulated annealing to find the optimal age, but all together, the brute force methodology works much better in finding the optimal claim age.