Final Project

Marc Eskew

5/30/2022

Problem Background

We are using modeling and simulation to identify an optimum distribution of shares between "high" and "low" risk options. With a few assumptions (no taxes, dividends, additional capital injections), we can set up a model for the price of a low risk asset as:

$$F(t) = e^r F(t-1)$$
$$st. r > 0$$

This low risk asset appreciates deterministicly as long as r is constant. Additionally there are no transaction costs for this asset.

The high risk asset is modeled as follows:

$$S(t) = S(t-1)R(t)$$
$$log(R(t)) = (\mu_0 + Z(t)\sigma_0)J(t) + (\mu_1 + Z(t)\sigma_1)(1 - J(t))$$

$$J(t) \sim Ber(p)$$
$$Z(t) \sim N(0,1)$$

With transactional costs modeled as $c = \rho a^2 S$ where a is the number of shares bought or sold and S is the price. This model gives you a return based on μ_0 and σ_0 with probability p and the 1 parameters with $1-p$. The RV $Z(t)$ determines the realization around μ scaled by σ.

With a budget of b, we are interested in determining the optimum allocation of shares between the high and low risk assets, θ, ϕ respectively, such that our expected utility, $u(x) = log(max(x,0))$ at t=1 is maximized.

Formulate the optimization Problem

As the low risk asset has a guarenteed rate of return, we can see that $max(x,0)$ will always be a positive value (we can just invest our entire budget using the risk free strategy). We can set up our maximization problem as such:

$$\max_{\theta(0),\phi(0)} log(\theta F(1) + \phi S(1) - \phi^2 \rho S(1))$$
$$st \theta F(0) + \phi S(0) + \phi^2 S(0)\rho = b$$

We can substitute the given values for $F(0), S(0), \rho, b$:

$$\max_{\theta(0),\phi(0)} log(\theta F(1) + \phi S(1) - \phi^2 \rho S(1))$$
$$st \theta + 2\phi + 0.02\phi^2 = 200$$

Rearraning the constraint to solve for θ, $\theta = -2\phi - 0.02\phi^2 + 200$ and substituting into the objective function, $log([-2\phi - 0.02\phi^2 + 200]F(1) + \phi S(1) - \phi^2 \rho S(1))$ will give us an unconstrained optimization problem. As we are trying to find the maximum (or negative minimum) of this function over a defined support, we can solve this by using the first derivative with respect to ϕ and determining the root.

$$\frac{d}{d\phi} log([-2\phi - 0.02\phi^2 + 200]F(1) + \phi S(1) - \phi^2 \rho S(1)) = \frac{F(1)(-0.04\phi - 2) - 0.02S(1)\phi + S(1)}{S(1)\phi(1 - 0.01\phi) + F(1)(200 - 2\phi - 0.02\phi^2)}$$

$F(1)$ can be calculated exactly, however, we can use simulation to sample $S(1)$ and determine our expected utility.

Simulation (2a and 2b)

First, we will set our constant values and functions.

```
f_0 <- 1
s_0 <- 2
p_val <- .4
mu_0 <- .1
sigma_0 <- .15
mu_1 <- .25
sigma_1 <- .3

rho <- .01
r_val <- .03
b_val <- 200


s <- function(s,r) {
  s*r
}

r_fun <- function(mu_0,sigma_0,mu_1,sigma_1,p){

  z <- rnorm(1)
  j <- as.numeric(rbernoulli(1, p = p))

  exp((mu_0+z*sigma_0)*j + (mu_1+z*sigma_1)*(1-j))
}

f_fun <- function(f,r){
  exp(r)*f
}
```

The following formula finds the root of the derivative, the optimum value for ϕ given $S(1)$. The root finding function, fzero, is similar to the function in MATLAB. It was returning values where the function approaches +/- infinity; some additional logic was added to ensure only the root at 0 was found. If no root was found in the support space, a value of 0 is returned.

```
applyfun <- function(sval,fval){

  df <- data.frame()
  for(i in seq(from =-100, to =100, by = 20)){
```

```
    rt1 <- tryCatch(pracma::fzero(function(x)
      {(sval + fval*(-2-.04*x) - .02 * sval * x) /
        (sval*x*(1-.01*x) + fval * (200 - 2*x - .02 *x^2))},
                                  x = i),
      error = function(e) list(x = 0, fval = 10e10))

    df <- bind_rows(df, data.frame(x=rt1$x,fval=rt1$fval))

  }
  val <- df[which.min(abs(df[,2])),1]

  if(val < 0 ) {
    0
  } else {
    val
  }
}
```

The final function takes the known constants, runs the simulation, and calculates the optimal value of ϕ and the expected utility within a designed error percentage and level of confidence.

```
fin_func <- function(s_0,p,mu_0,sigma_0,mu_1,sigma_1,rho,r,b,
                     gamma,delta,n_0,N_0) {

  meanv <-0
  mean_ux <- 0
  mmt <- 0
  mmt_ux <- 0
  eps <- gamma
  i <- 0
  max_phi <- 0
  f_1 <- f_fun(1,r)
  err <- Inf
  out_val <- data.frame()

  while(i < N_0 | err >= eps){

    s1 <- s(s_0,r_fun(mu_0,sigma_0,mu_1,sigma_1,p))

    gz <- applyfun(s1,f_1)
    ux <- log(gz*s1 + ((b-s1*s_0))*f_1)

    mean_ux <- (mean_ux * i + ux) / (i+1)
    mmt_ux <- (mmt_ux*i+ux^2)/(i+1)
    std_ux = sqrt(mmt_ux- mean_ux^2)
    err_ux <- qnorm(1-gamma/2)*std_ux/sqrt(i)

    meanv <- (meanv * i + gz) / (i+1)
    mmt <- (mmt*i+gz^2)/(i+1)
    std = sqrt(mmt- meanv^2)

    i <- i+1

    eps <- delta*meanv
```

```
    err <- qnorm(1-gamma/2)*std/sqrt(i)
    out_val <- rbind(out_val,data.frame(i = i, g = meanv,g_err = err,
                                        ux = mean_ux,ux_err = err_ux,
                                        max = max_phi,gz = gz,s = s1))
  }

  list(data=out_val,theta_mean = meanv, theta_error = err,
       u_mean = mean_ux, u_error = err_ux)
}
```

Using this we can execute a simulation to get our parameters of interest.

```
df<- fin_func(2,p_val,mu_0,sigma_0,mu_1,sigma_1,rho,r_val,b_val,.05,.05,10,100)
tail(df$data)
```

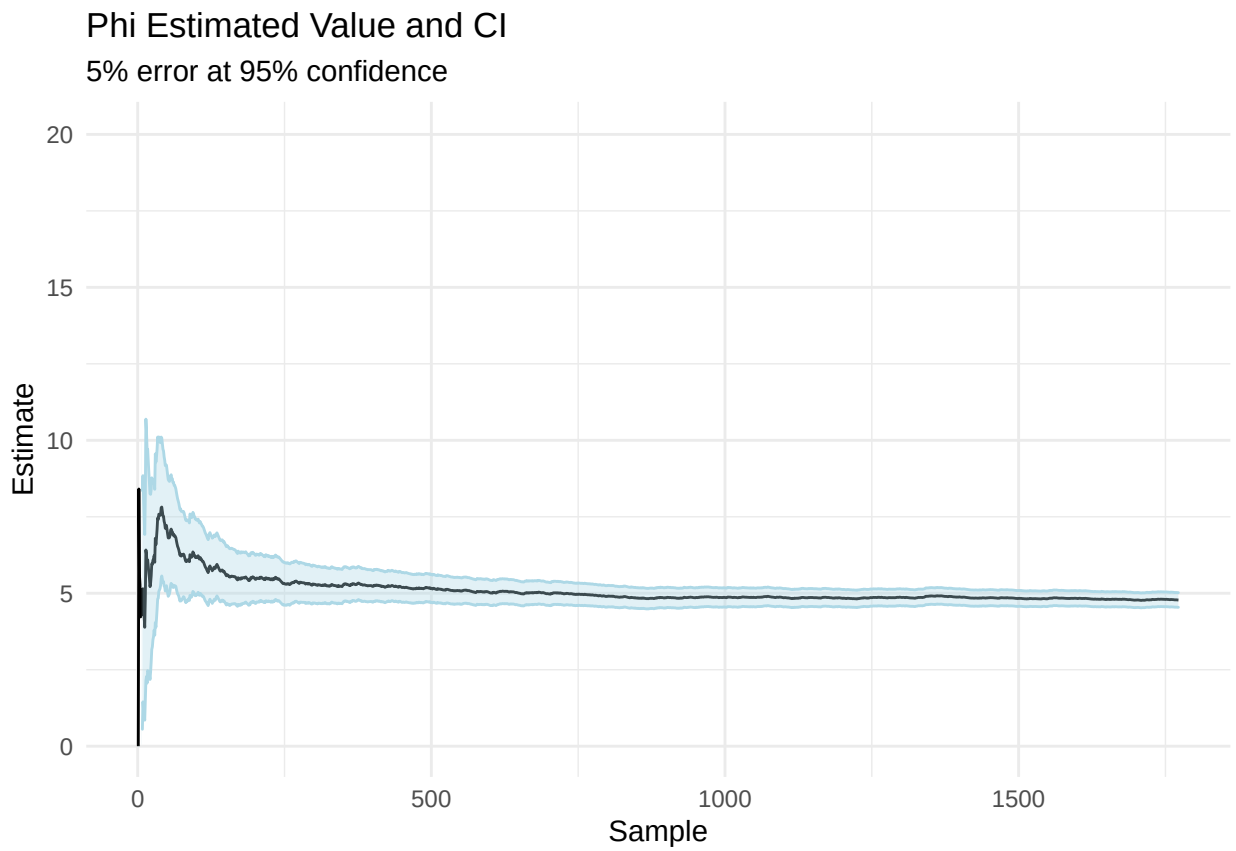
```
##         i        g     g_err       ux      ux_err max        gz        s
## 1767 1767 4.783736 0.2394948 5.373508 0.003919396   0 3.8463360 2.404412
## 1768 1768 4.786223 0.2394090 5.373538 0.003917620   0 9.1813577 2.988032
## 1769 1769 4.783800 0.2393208 5.373504 0.003915993   0 0.4998879 2.102534
## 1770 1770 4.781804 0.2392175 5.373473 0.003914232   0 1.2511138 2.166693
## 1771 1771 4.780899 0.2390890 5.373455 0.003912187   0 3.1790017 2.340768
## 1772 1772 4.780063 0.2389597 5.373437 0.003910130   0 3.2984891 2.352030
```

```
df[2:5]
```

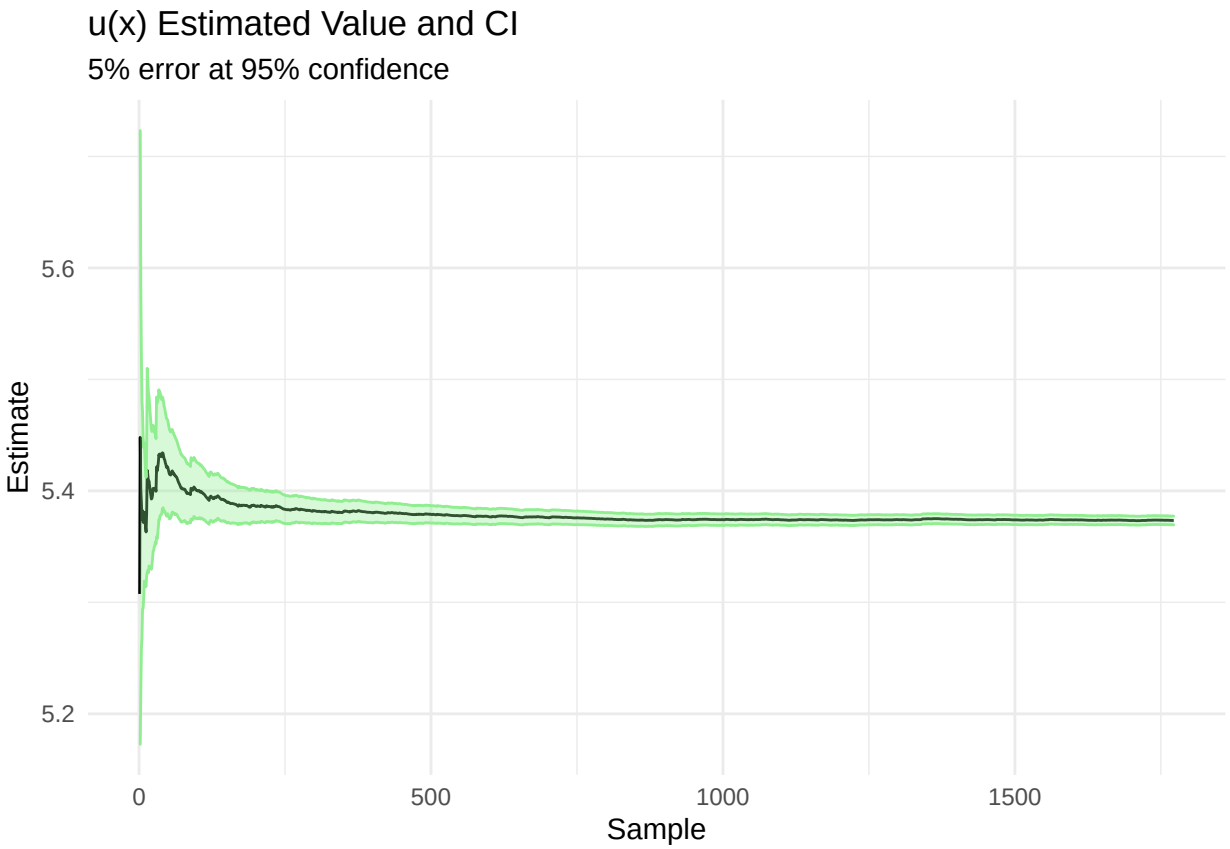
```
## $theta_mean
## [1] 4.780063
##
## $theta_error
## [1] 0.2389597
##
## $u_mean
## [1] 5.373437
##
## $u_error
## [1] 0.00391013
```

With the supplied constants, we can predict that the optimal share of risk free assets to purchase, ϕ, has a mean value of 4.7800626 and the true value is within 5% error, or between 4.5411029 and 5.0190223, with 95% confidence. Additionally, $u(x)$ was calculated along with the samples for ϕ and the variance is lower, we can state that $E(u(x)) = 5.3734375$ within 5% error at 95% confidence (5.3695273,5.3773476). This estimate is justified due to the law of large numbers stating that as the number of samples increase, the closer the estimator gets to the true value.

```
ggplot(df$data) +
  geom_path(aes(x = i, y = g)) +
  geom_ribbon(aes(x = i, ymin = g-g_err, ymax = g+g_err),
              fill = 'lightblue',color="lightblue", alpha = .35) +
  theme_minimal() +
  scale_y_continuous(limits = c(0,max(df$data$g+df$data$g_err))) +
  labs(
    title = "Phi Estimated Value and CI",
    subtitle = "5% error at 95% confidence",
    x = "Sample",
    y = "Estimate"
  )
```




```
ggplot(df$data) +
  geom_path(aes(x = i, y = ux)) +
  geom_ribbon(aes(x = i, ymin = ux-ux_err, ymax = ux+ux_err),
              fill = 'lightgreen',color="lightgreen", alpha = .35) +
  theme_minimal() +
  labs(
    title = "u(x) Estimated Value and CI",
    subtitle = "5% error at 95% confidence",
    x = "Sample",
    y = "Estimate"
  )
```



Variance Reduction

In order to sample more efficiently, a variance reduction technique was implemented. The technique used was bias reduction and the "cool estimator" to improve our sampling process. This process was used in order to eliminate the bias present in the sampling, lower the variance between draws, and improve the performance of the process.

```
fin_func2 <- function(s_0,p,mu_0,sigma_0,mu_1,sigma_1,rho,r,b,gamma,delta,n_0,N_0) {

  meanv <-0
  mmt <- 0
  mmt_ux <- 0
  eps <- gamma
  i <- 0
  f_1 <- f_fun(1,r)
  mean_ux <- 0
  max_phi <- 0
  err <- Inf
  pg <- 1-(.5)^(3/2)
  out_val <- data.frame()

  while(i < 10 | err >= eps){


    N <- rgeom(1,pg) + n_0
```

```
    s_1 <- replicate(2^(N+1),s(s_0,r_fun(mu_0,sigma_0,mu_1,sigma_1,p_val)))

    val <- sapply(s_1,applyfun,fval = f_1)

    val_even <- val[c(TRUE,FALSE)]
    val_odd <- val[c(FALSE,TRUE)]

    gz <- (mean(val) - (mean(val_odd)+mean(val_even))/2)/(pg*(1-pg)^(N-n_0)) + mean(val[1:2^n_0])

    ux <- log(val*s_1 + ((200-s_1*2))*f_1)
    ux_even <- ux[c(TRUE,FALSE)]
    ux_odd <- ux[c(FALSE,TRUE)]

    ux <- (mean(ux) - (mean(ux_odd)+mean(ux_even))/2)/(pg*(1-pg)^(N-n_0)) + mean(ux[1:2^n_0])


    meanv <- (meanv * i + gz) / (i+1)
    mmt <- (mmt*i+gz^2)/(i+1)
    std = sqrt(mmt- meanv^2)

    mean_ux <- (mean_ux * i + ux) / (i+1)
    mmt_ux <- (mmt_ux*i+ux^2)/(i+1)
    std_ux = sqrt(mmt_ux- mean_ux^2)


    i <- i+1


    eps <- delta*meanv
    err <- qnorm(1-gamma/2)*std/sqrt(i)
    err_ux <- qnorm(1-gamma/2)*std_ux/sqrt(i)

    out_val <- rbind(out_val,data.frame(i = i, g = meanv,g_err = err,
                                        ux = mean_ux,ux_err = err_ux,
                                        max = max_phi,gz = gz))

  }
  list(data=out_val,theta_mean = meanv, theta_error = err, u_mean = mean_ux, u_error = err_ux)
}
```

We can run this function with the same input parameters, with the addition of n_0 to determine the burn in period of the estimator.

```
df1 <- fin_func2(2,p_val,mu_0,sigma_0,mu_1,sigma_1,rho,r_val,b_val,.05,.05,5,100)

tail(df1$data)
```

```
##     i        g     g_err       ux      ux_err max       gz
## 57 57 4.747653 0.2510755 5.373120 0.004228678   0 5.473858
## 58 58 4.760232 0.2479542 5.373286 0.004168176   0 5.477255
## 59 59 4.773518 0.2451151 5.373507 0.004119922   0 5.544103
## 60 60 4.790181 0.2431958 5.373709 0.004070359   0 5.773293
## 61 61 4.807603 0.2415943 5.373958 0.004032808   0 5.852916
## 62 62 4.827881 0.2409445 5.374223 0.004001032   0 6.064833
```

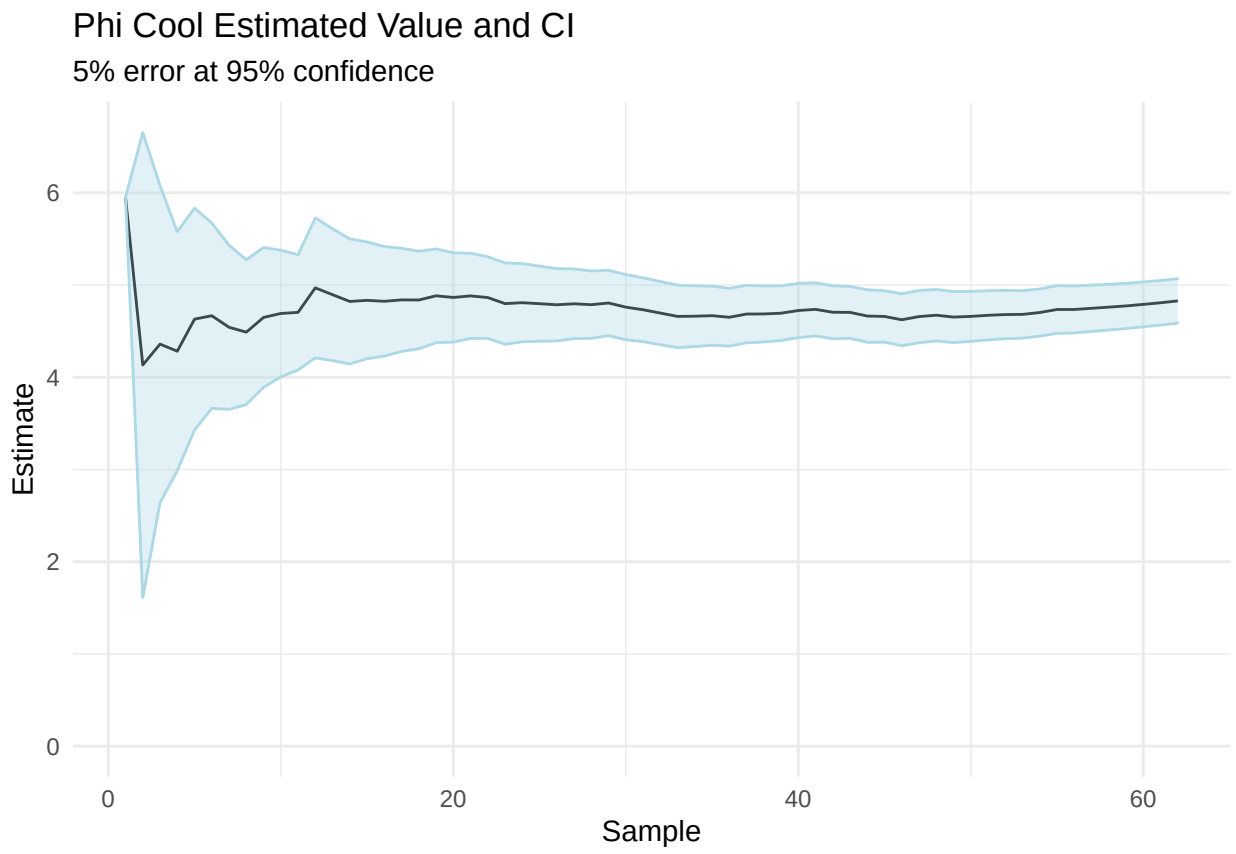
```
df1[2:5]
```

```
## $theta_mean
## [1] 4.827881
##
## $theta_error
## [1] 0.2409445
##
## $u_mean
## [1] 5.374223
##
## $u_error
## [1] 0.004001032
```

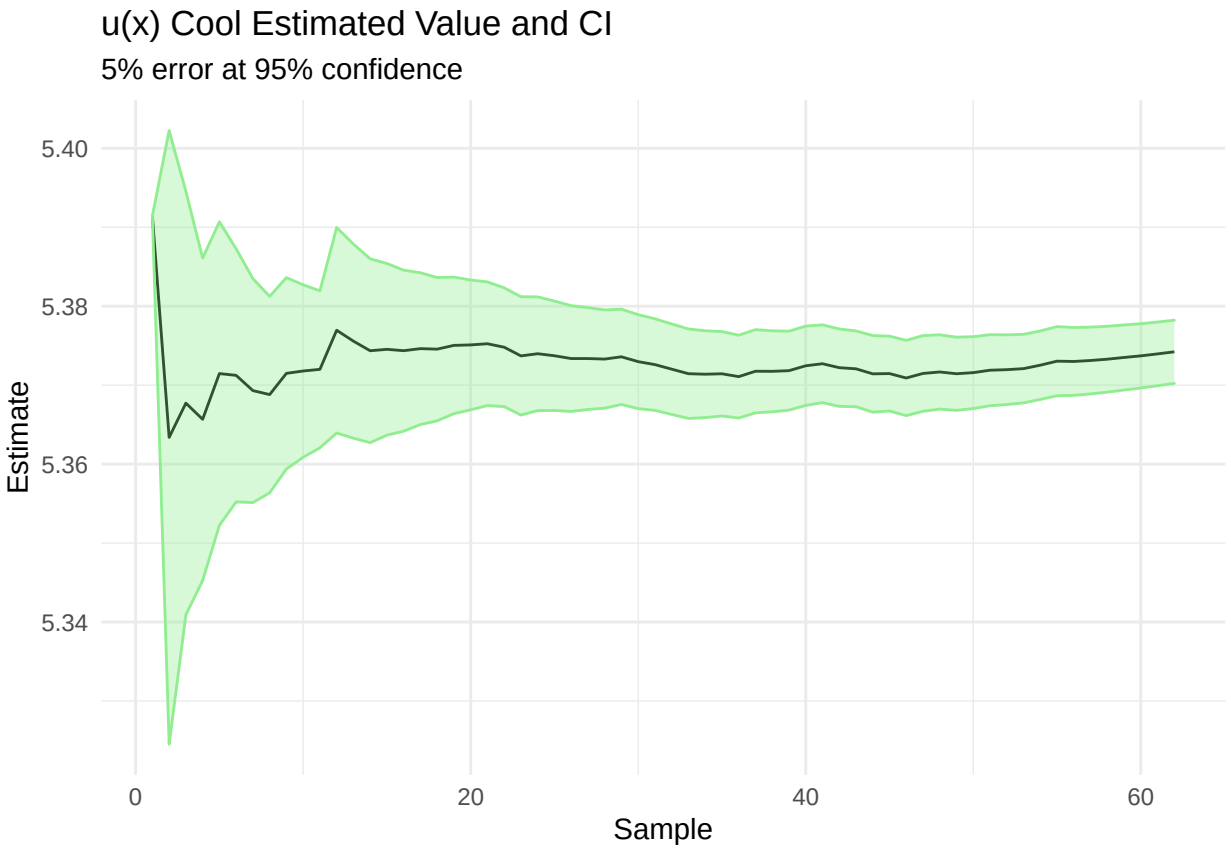
Using variance reduction we can attain a value within our desire error limits with less iterations. We find that ϕ has a mean value of 4.8278809 and the true value is within 5% error, or between 4.5869364 and 5.0688255, with 95% confidence. Additionally, $u(x)$ was calculated along with the samples for ϕ and the variance is lower, we can state that $E(u(x)) = 5.3742232$ within 5% error at 95% confidence (5.3702221,5.3782242).

Plots generated from the cool estimators will show reduced variance and less iterations until convergence.

```
ggplot(df1$data) +
  geom_path(aes(x = i, y = g)) +
  geom_ribbon(aes(x = i, ymin = g-g_err, ymax = g+g_err),
              fill = 'lightblue',color="lightblue", alpha = .35) +
  theme_minimal() +
  scale_y_continuous(limits = c(0,max(df1$data$g+df1$data$g_err))) +
  labs(
    title = "Phi Cool Estimated Value and CI",
    subtitle = "5% error at 95% confidence",
    x = "Sample",
    y = "Estimate"
  )
```




```
ggplot(df1$data) +
  geom_path(aes(x = i, y = ux)) +
  geom_ribbon(aes(x = i, ymin = ux-ux_err, ymax = ux+ux_err),
              fill = 'lightgreen',color="lightgreen", alpha = .35) +
  theme_minimal() +
  labs(
    title = "u(x) Cool Estimated Value and CI",
    subtitle = "5% error at 95% confidence",
    x = "Sample",
    y = "Estimate"
  )
```



I used this technique because it seemed the most straight forward to implement given this problem. Determining a control variate would have been difficult as it would require development of a correlated CV. Utilizing conditional Monte Carlo, likewise, would have been challenged to determine the explicit conditional probabilities.

Conclusion

Utilizing the developed financial model and Monte Carlo estimation, we've shown that it is possible to determine and optimal value to maximize a utility function based on return. This function is fully parameterized and can accept different values as market conditions change. By utilizing the values given to us for this scenario, we are able to make a recommendation to purchase approximately 4.7800626 shares to maximize expected return.

I completed the project by myself, no group work was conducted.