Model Write Up

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

The model has a structure in which agents may live for 2 time periods. In the first period, they are young and save for retirement. In the second period they are old and dissave.

The initial generation of the model begins as Young, and each agent of the generation is assigned a wage type as either "rich" or "poor". Rich agents earn a higher wage than poor agents, and this wage data is stored as a vector.

```
#population counts
rich_population <- 5000
poor_population <- 10000
#initial wages
rich_wage <- 4
poor_wage <- 2

# Initialize wages vector
wages <-c(rep(rich_wage,rich_population), rep(poor_wage, poor_population))
#Initialize retired vector (empty)
retired <- c(rep(0, rich_population+poor_population))
```

Note, that during the first period some agents will have died (see random death age and savings). Their life savings is a function of death age such that if they died before retirement

age, life savings will be proportionally less. However, in case of premature death their life savings will continue to be stored with the rest of the generational cohort being saved for the next generation. At the first time step, wages are converted to a savings amount. Now, the young generation becomes old and enters retirement with their life savings.

```
savings_rate <- 0.5
#Initialize life savings vector
working <- wages * savings_rate
```

Meanwhile, a new young generation enters the workforce. They have no savings, but each agent does "inherit" a wage type from their parent agent. Their wage type is stochastically determined, with a preference toward their parent's type.

Computationally, this is achieved by first identifying the parent's type, and generating a random probability from the uniform distribution of 0 to 1, $U \sim \text{Unif}(0, 1)$ if $U < 1 - p_i$, where p_i is the probability of moving from class i, the agent receives the wage type of their parents. If not, they switch wage types.

```
prob_rich_to_poor <- 0.1 #a 10% chance of a child from a rich parent becoming poor
prob_poor_to_rich <- 0.05 #5% chance of a child of a poor parent becoming rich


#class movement
update_class <- function(wages) {
  new_wages <- lapply(wages, function(income) {
    if (income == poor_wage) {
      if (runif(1) < (1-prob_poor_to_rich)) {
        return(poor_wage)
      }
      else {
        return(rich_wage)
      }
    }
    else {
      if (runif(1) < (1-prob_rich_to_poor)) {
        return(rich_wage)
      } else {
        return(poor_wage)
      }
    }
  })
```

```
  return(unlist(new_wages))
}
```

Traditionally, calculation of wage rates in OLG models is dependent on the Cobb-Douglas Production function, or Solow Growth model. This takes capital and labor as inputs for aggregate output.

$$Y = K^{\alpha} L^{(1-\alpha)}$$

The marginal product of labor (wage rate) is found by differentiating output with respect to labor.

$$MPL = (1-\alpha)(Y)/L$$

This approach works well in homogeneous models, but the singular wage rate has constraints. When introducing class heterogeneity, this approach needs to be altered.

For now, we'll assume that different wages are a result of different marginal products of labor. Rich workers might have higher productivity as a result of education or skill differences. Therefore, we can say there are two types of labor contributing the the economy, rich labor L_r and poor labor L_p. A new production function can be defined

$$Y = K^{\alpha}(L_r^{\beta} L_p^{(\gamma)})$$

with the condition that $\alpha+\beta+\gamma = 1$ in order to maintain constant returns to scale. Note that labor types are combined multiplicatively here, implying these two types of labor complement each other. That is, both types of labor are necessary for the economy to function. Indeed, the productivity of many high-wage workers (e.g. executives) is dependent on the labor of lower-wage workers (e.g. administrative staff).

Additionally, recall that exponents α, β, γ represent responsiveness of aggregate output to the supply of their relative inputs. Each parameter her represents the percent change in output as the result of a 1% increase in an input. Parameter values of $\alpha = 0.3, \beta = 0.4, \gamma = 0.3$ indicate that output grows 0.3% from a 1% increase in capital, 0.4% from a 1% increase in wigh-wage labor, and 0.3% from a 1% increase in low wage labor. By definition, parameters β and γ represent a marginal product of their labor type. The greater the difference between β and γ, the greater the difference in productivity, and the greater the difference in wages.

As in the case of homogeneity, marginal products of labor are derived by differentiating.

$$MPL_r = \frac{\partial Y}{\partial L_r} = \beta K^{\alpha} L_r^{(1-\beta)} L_p^{\gamma}$$

$$MPL_p = \frac{\partial Y}{\partial L_p} = \gamma K^{\alpha} L_r^{\beta} L_p^{(1-\gamma)}$$

```
### Cobb-Douglas parameters, must sum to 1

alpha <- 0.3
beta <- 0.5
gamma <- 0.2

#function to compute total capital
compute_capital <- function(retired) {

  # Retired agents wealth is their savings
  retired_capital <- sum(retired)

  # Total capital in the economy is total savings b/c no foreign investment
  total_capital <- retired_capital
  return(total_capital)
}

# Function to compute the number of workers
total_labor <- function(working) {
  return(length(working))  # The number of working agents
}

# Function to calculate the marginal product of labor (wage) using the Cobb-Douglas productio
update_wages <- function(wages) {
  #separating working agents into ricch and poor groups
  rich_agents <- wages[wages == rich_wage]  # Agents earning the rich wage
  poor_agents <- wages[wages == poor_wage]  # Agents earning the poor wage


  # Compute total capital (wealth)
  total_capital <- compute_capital(retired)

  # Total labor (number of working agents)
  rich_labor <- length(rich_agents)
  poor_labor <- length(poor_agents)

  # Cobb-Douglas production function for the marginal product of labor (MPL)
  total_output <- total_capital^alpha * rich_labor^beta * poor_labor^gamma  # Total output i
```

```
  new_rich_wage <- beta * total_output * rich_labor^(1-beta) * poor_labor^gamma
  new_poor_wage <- gamma * total_output * rich_labor^beta * poor_labor^(1-gamma)
  return(c(new_rich_wage,new_poor_wage))
}
```

After solving for rich and poor wages, each agent is assigned a wage in accordance to their type.

```
update_savings <- function(rich_wage,poor_wage, new_rich_wage, new_poor_wage, wages){
  new_working <-lapply(wages, function(income) {
    if (income == poor_wage) {
      return(new_poor_wage)
    }
    else {
      return(new_rich_wage)
    }
  })

  return(unlist(new_working))
}
```

The following time step will see these wages manifest as life savings in the same manner as the first generation. However, this generation receives an addition to their life savings, which is a bequest from their parents. The bequest is a percentage of each agent's parent's life savings determined as a function of their wage type and death age.

Modeled bequests her are built on the assumption that bequests are largely accidental. That is, an agent consumes with the plan to spend all of their wealth over their lifetime. In this case, inheritance is simply the result of unplanned early death.

Using the 2021 Actuarial Table from the Social Security Administration, we are able to find death by age data.

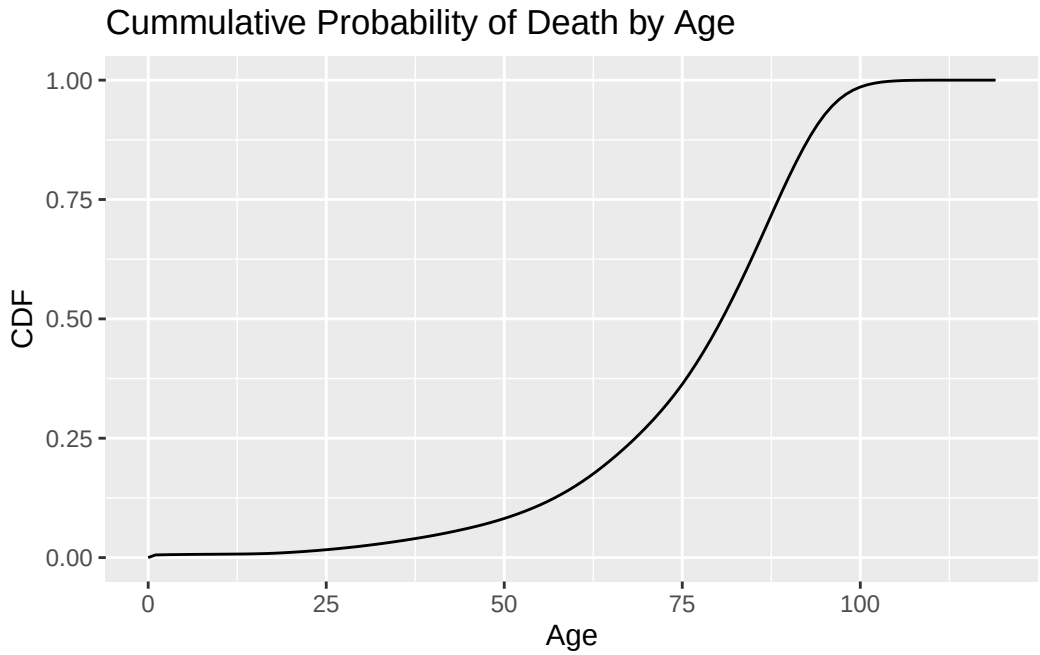
```
Social_Security_Deaths_2021 <- read_csv(".github/workflows/Social Security Deaths 2021.csv")
```

```
Rows: 120 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): age, total births, total deaths
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We can normalize this so that total deaths is a cummulative probability distribution of death from ages 0 to 119.

```
cdf_data <- Social_Security_Deaths_2021 |>
  mutate(cdf = `total deaths`/205000)

ggplot(cdf_data, aes(x = age, y = cdf))+
         geom_line()+
  labs(title = "Cummulative Probability of Death by Age",
       x = "Age",
       y = "CDF")
```



It is possible to generate data according to a cumulative distribution function through a process known as inverse transform sampling.

We generate $U \sim$ Unif$(0, 1)$ which is a random number between 0 and 1 and acts as kind of target probability.To generate a random age in the set, we choose the least X such that $F(X) \leq U$. That is, we choose the greatest age X such that the probability of a random death occuring before X is less than or equal to the uniform probability generation. With this method, any percentile of age at death is equally likely, but actual age at deaths are exactly as likely as the original distribution.

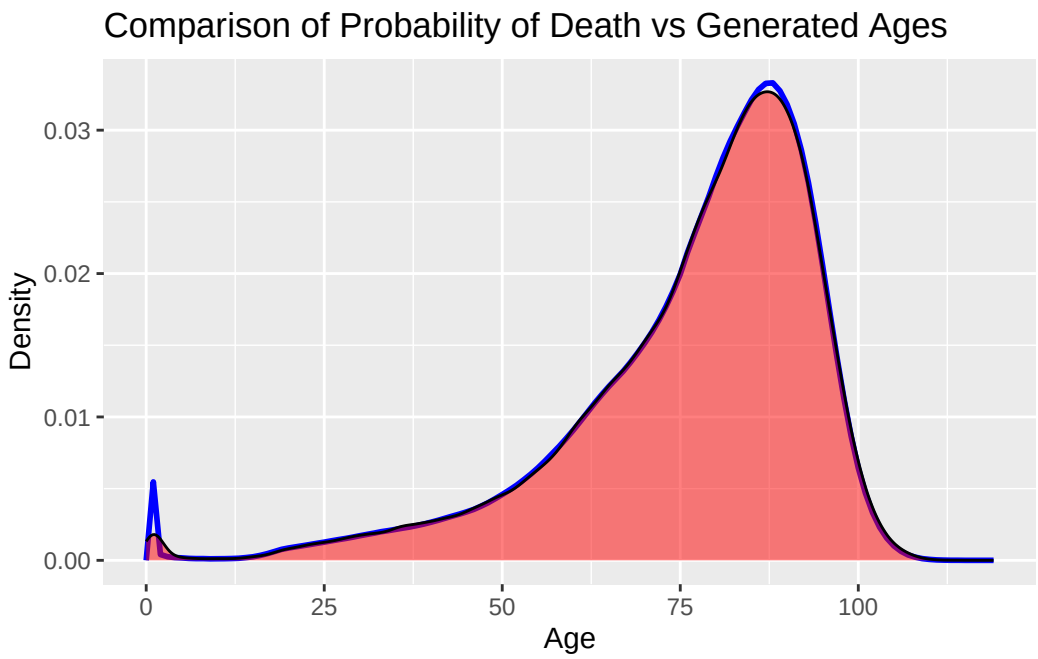
```r
# Function for inverse transform sampling
inverse_transform_sampling <- function(cdf_data, n_samples) {
  # Step 1: Generate `n_samples` uniform random numbers between 0 and 1
  uniform_randoms <- runif(n_samples)

  # Step 2: For each random number, find the corresponding value in the CDF
  sampled_values <- sapply(uniform_randoms, function(u) {
    # Find the smallest `age` where `cdf >= u`
    age_sampled <- cdf_data$age[min(which(cdf_data$cdf >= u))]
    return(age_sampled)
  })

  return(sampled_values)
}

death_age <- inverse_transform_sampling(cdf_data, rich_population+poor_population)
```

Below, see that 100,000 death ages have been generated using this method. We can compare their density (in red) to the true probability density of the actuarial table (in blue). Unlike the cumulative distribution, this shows probability of death at any single age.

```r
set.seed(4747)
n_samples <- 100000
random_samples <- inverse_transform_sampling(cdf_data, n_samples)


true_prob_deaths <- Social_Security_Deaths_2021 |>
  mutate(deaths_at_age = `total deaths` - lag(`total deaths`, default = first(`total deaths`)
  mutate(prob_of_death = deaths_at_age/205000)


# Create a data frame of generated ages for histogram
generated_ages_df <- data.frame(age = random_samples)

# Plot
ggplot() +
  #pdf (line)
  geom_line(data = true_prob_deaths, aes(x = age, y = prob_of_death), color = "blue", size =

  #generated ages (density)
  geom_density(data = generated_ages_df, aes(x = age), fill = "red", alpha = 0.5) +
```

```
  labs(title = "Comparison of Probability of Death vs Generated Ages",
       x = "Age",
       y = "Density")
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



To apply this to the model, we can create a vector representing every individual's death age determined by inverse transform sampling of the Social Security distribution.

```
death_age <- inverse_transform_sampling(cdf_data, rich_population+poor_population)
```

We transform this vector with a piecewise function of death age to obtain what fraction of total potential savings the individual leaves behind for the next generation.

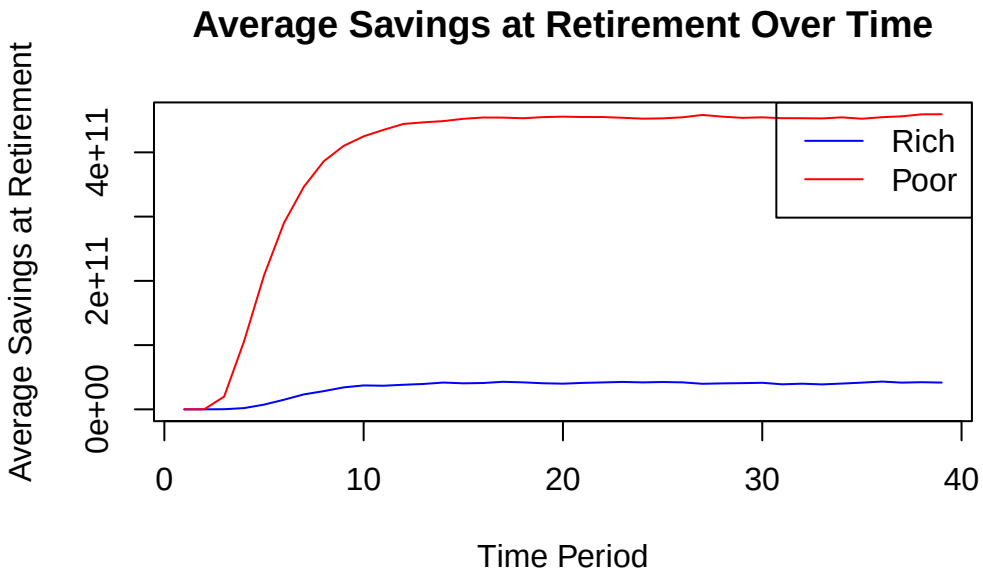
```
#piecewise bequest rate definition:
#if death age (da) <= 20, br = 0
#if 20< da < 60, br = 2.5 (da/100) - 0.5
#if 60< da, br = -2.5(da/100) + 0.5

br <- ifelse(death_age < 20, 0,
```

```
              ifelse(death_age < 60, 2.5 * (death_age / 100) - 0.5,
                     ifelse(death_age < 100, -2.5 * (death_age / 100) + 2.5, 0)
              )
)
```

With that, one full generation cycle is complete. To model this, we call the outlined steps and functions through many iterations to find equilibrium.

```
time_periods <- 40

av_savings <- function(wages, retired) {
  rich_agents <- retired[wages == rich_wage]  # Agents earning the rich wage
  poor_agents <- retired[wages == poor_wage]  # Agents earning the poor wage
  rich_savings <- mean(rich_agents)
  poor_savings <- mean(poor_agents)
  return(c(rich_savings, poor_savings))
}

# Initialize vectors to store average savings over time
rich_avg_savings <- c()
poor_avg_savings <- c()

tracking_wages_rich <- c()
tracking_wages_poor <- c()

############ Main simulation loop#############################
for (t in 1:(time_periods - 1)) {
  #print(paste("Time Period:", t))
  #print("Working List:")
  #print(working)
  #print("Retired List:")
  #print(retired)


  # Compute capital and labor
  #total_capital <- compute_capital(working, retired)
  #labor <- total_labor(working)

  #Updated Bequest Rate
  bequest_rate <- ifelse(death_age < 20, 0,
                ifelse(death_age < 60, 2.5 * (death_age / 100) - 0.5,
                       ifelse(death_age < 100, -2.5 * (death_age / 100) + 2.5, 0)
```

```
              )
  )
  #Updated death ages for new generation
  death_age <- inverse_transform_sampling(cdf_data, rich_population+poor_population)

  # Update the retired list with new retirees and their inheritance
  retired <- bequest_rate*retired + working


  #for graph
  avg_savings <- av_savings(wages,retired)
  rich_avg_savings <- c(rich_avg_savings, avg_savings[1])
  poor_avg_savings <- c(poor_avg_savings, avg_savings[2])
  tracking_wages_rich <- c(tracking_wages_rich, rich_wage)
  tracking_wages_poor <- c(tracking_wages_poor, poor_wage)

  #get new wage rates
  new_rich_wage <- update_wages(wages)[1]
  new_poor_wage <- update_wages(wages)[2]

  # Update the working list with class movement of new generation
  wages <- update_class(wages)
  #update working incomes
  wages <-update_savings(rich_wage, poor_wage, new_rich_wage,new_poor_wage, wages)
  working <- wages *savings_rate
  rich_wage <- new_rich_wage
  poor_wage <- new_poor_wage


}


############ Graph ##################

# Plotting the average savings over time
time_periods_plot <- 1:(time_periods - 1)

plot(time_periods_plot, rich_avg_savings, type = "l", col = "blue", ylim = range(c(rich_avg_s
     xlab = "Time Period", ylab = "Average Savings at Retirement",
     main = "Average Savings at Retirement Over Time")
lines(time_periods_plot, poor_avg_savings, col = "red")
legend("topright", legend = c("Rich", "Poor"), col = c("blue", "red"), lty = 1)
```


Average Savings at Retirement Over Time
Average Savings at Retirement
0e+00
2e+11
4e+11
0
10
20
30
40
Time Period
Rich
Poor