MODELING THE EFFECT OF POPULATION DYNAMICS WITH THE SOLOW GROWTH MODEL

A CHERPESKI, S MCINTYRE, A MELO, T SANDERS, AND M STAFFORD

Abstract. We investigated how various models of population dynamics affect the long term growth of a country. We modeled income growth using the Solow Growth Model augmented with an SIR-inspired population dynamics model. We investigated the model's response to various phenomena including heterogeneous worker productivity, exogenous shocks, saving habits, and immigration. The model exhibited robust and realistic responses to these phenomena. Ultimately, this analysis proves useful as a first step in guiding policymakers in forecasting how these phenomena may affect long term changes in income.

1. Background and Movitation

Understanding how a country's economy responds to factors including changes in capital, technology, shifting population dynamics, shocks, productivity, and various other factors is vital for understanding the effects of public policy decisions. In this project, we modeled GDP growth using the Solow Growth model combined with an SIR population growth model and evaluated its response to various modifications. The Solow Growth model was first developed by economist Robert Solow in 1956 [1]. The Solow model has the advantage of being comparatively simple and stylistically accurate, especially for industrialized economies. Other models such as the NGM or DSGM are more flexible but are beyond the scope of this project.

In most instances of direct modeling, population is assumed to be constant or to grow at a particular rate. However, this is not true in general as birth rates may change quickly in a given country. Indeed, in the last several decades, declining birth rates across the world have led to lots of questions about the effects that aging and shrinking populations have upon economic outlooks. For instance, the fertility rate in the US and India were 1.6 and 2.0 in 2021 (the last available data), but twenty years ago in 2001 were 2.0 and 3.3 - a sizeable shift for both countries [2]. That makes being able to forecast the effects of changing population dynamics increasingly important in determining things like the future of social security, retirement age, incentives for children, and the long term economic outlook of a country.

It is insufficient to measure a population with a single number since each age group is on average different from the others. For this reason, we use an SIR model for population that allows us to take a more refined view of how population changes affect variables of interest.

The Solow Growth model is defined as follows:

$$c(t) + i(t) = y(t)$$
$$sy(t) = i(t)$$
$$y(t) = A(t)k(t)^{\alpha}$$
$$\dot{k}(t) = sy(t) - (\delta + \Delta\bar{n}(t))k(t)$$

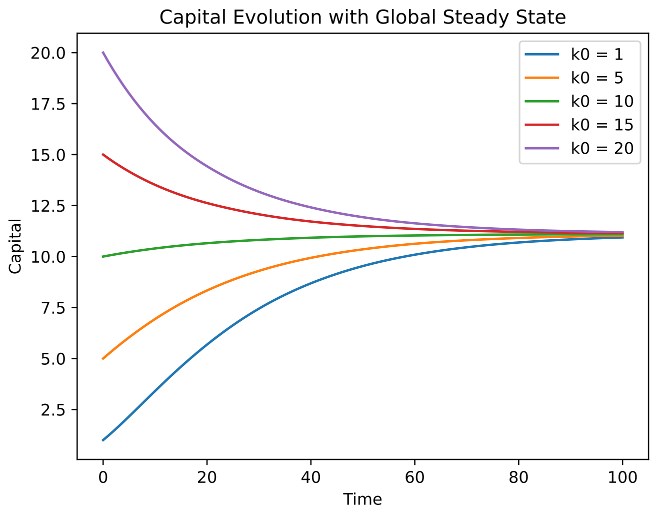
Where $c(t), i(t), y(t), k(t)$ are per person quantities of consumption, investment, income and capital. $A(t)$ is called TFP - total factor productivity, and can be used to model technology and other economy-wide inputs to productivity. The parameter α represents the effectiveness of capital in creating income; estimates in the US are typically about 0.4 – indicating that productivity is concave in capital per person. δ is a capital depreciation rate, representing how fast equipment needs to be replaced or skills refreshed; estimates vary but we used 0.08. s is the savings rate, which determines what proportion of income is invested, while $(1 - s)$ is the fraction of income that is consumed. Finally, $\Delta\bar{n}$ is a function representing the population growth rate, a parameter that will be replaced by our SIR population model.

This model is homogeneous because every worker in the economy is identical. Moreover, growth is completely determined by the chosen parameters and starting conditions. Capital growth is positively related to investment in new capital and negatively related to capital depreciation and population increase (since each quantity is modeled in per person terms). Departures from this homogeneous model will be explored in later sections.

Capital can be thought of as anything that makes an individual worker in an economy productive. For instance, tractors, computers, technical training, roads, etc. Worker productivity (and in this model, their income) is determined by $A(t)k(t)^{\alpha}$, with $A(t)$ representing how effectively capital is used in the whole economy. It is important that productivity be concave in capital, this represents the fact that the first units of capital result in more meaningful increases in productivity than later units - a farmer's first tractor is instrumental, the eighth one is less so.

Unlike more complex models, the SGM model does not support consumer choice, so units of consumption are not as meaningful here. In general however, individuals in an economy will want to maximize the level of consumption per person as opposed to income per person. For instance, an economy that saves everything achieves a high value of income per person, but no one consumes anything - that country would be unhappy. Conversely, a country which saves nothing would run out of capital, leading to a loss of both consumption and income. However, in this paper we largely pass over questions of consumption versus income and focus mainly on the latter.

The SGM has two steady states, a trivial one at $k(t) = 0$ and the globally stable steady state we are interested in studying. The globally stable steady state occurs when $\dot{k} = 0$, meaning $\bar{k} = \frac{y(t)}{(\delta+\Delta\bar{n})}$, or $\bar{k} = \frac{sA(t)}{(\delta+\Delta\bar{n})^{\frac{1}{1-\alpha}}}$. That the steady state is globally stable can be derived analytically and is illustrated pictorially below.



2. Modeling

For the population model, we took inspiration from the SIR Model. In our population model, each age was discretized and treated as a state with individuals transitioning from one age to the next at each time step. At each time step, new people are born according to a given fertility rate and number of people of childbearing age, and individuals die at a fixed age according to life expectancy. In order to allow for population growth, we also removed the population normalization typical of SIR models. We introduced the following system to model population dynamics over time:

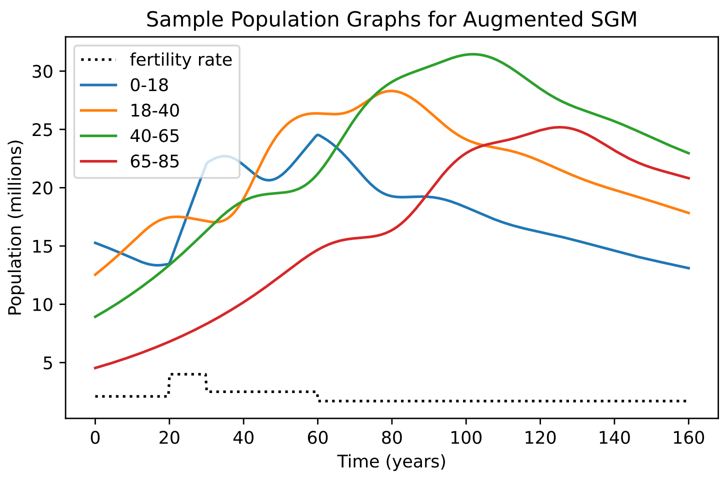
$$\dot{n}_0(t) = \frac{\gamma}{2(b_f - a_f + 1)} \sum_{i=a_f}^{b_f} n_i(t) \ - n_0(t)$$

$$\dot{n}_i(t) = n_{i-1}(t) - n_i(t), \ \forall i \in \{1, ..., E\}$$

Where n_i equals the number of people who are i years old, γ represents the current fertility rate (the number of children a woman has on average–a common way to measure births in a population), $\{a_f, a_f + 1, ...b_f\}$ are the ages where women can have children, and E is the average life expectancy of individuals. When the fertility rate is above 2, the population trends exponentially to infinity. When it is below 2, it trends to 0. In the event that the fertility rate equals 2 exactly, it has a finite, positive equilibrium.

For this model, we assumed that everyone lives to the age of life expectancy exactly, ignoring premature deaths and people who live past the life expectancy. We also assumed that the population is evenly divided between men and women, which is why we divided the first term by two. Combining this population equation with the prior economic model yields the following system of equations:

$$\dot{n}_0(t) = \tfrac{\gamma}{2(b_f - a_f + 1)} \sum_{i=a_f}^{b_f} n_i(t) \ - n_0(t)$$
$$\dot{n}_i(t) = n_{i-1}(t) - n_i(t), \ \forall i \in \{1, ..., E\}$$
$$\dot{k}(t) = sy(t) - (\delta + \Delta\bar{n}(t))k(t),$$
$$\Delta\bar{n}(t) = \frac{n_{a_{f-1}}(t) - n_{b_f}(t)}{\sum_{j=a_f}^{b_f} n_j(t)}$$



Here we include an example of the Augmented SGM. Notice how the decline in birth rates has a lagged effect both on population and on GDP. In the following section, we will devote ourselves to examining additional variations and modifications to this base model. These will include external shocks, changes in worker productivity, and immigration.

3. Results and Analysis

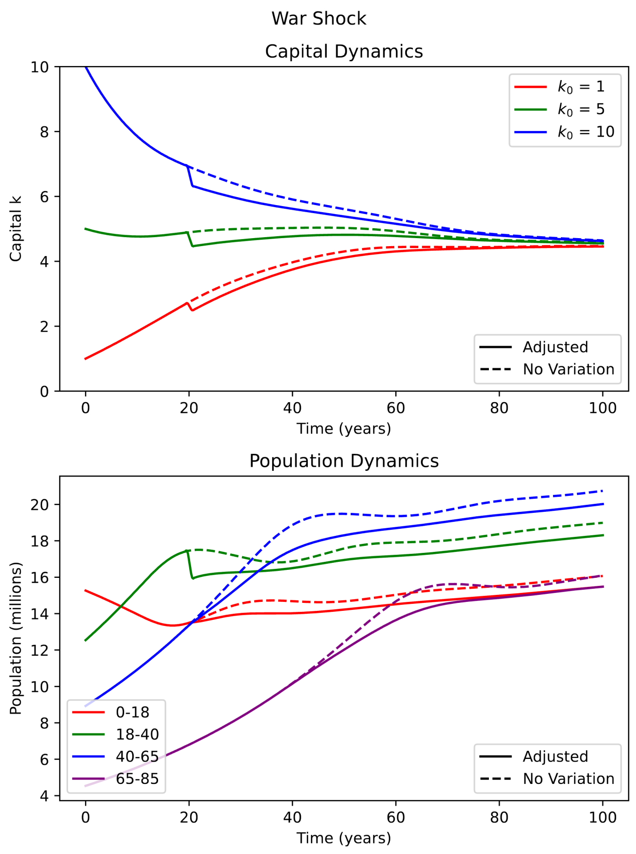
A shock to an ODE is the presence of a sudden perturbation to the system. An economy might experience a shock for a number of reasons. We investigated the effects of the shocks of war and technology investment.

3.1. **War.** While wars tend to have wide ranging effects on nearly every facet of the economy, we assumed that the primary effects are sudden loss of capital and sudden loss of life. Specifically, we modeled this war as an instantaneous loss of 10% of capital and 10% of the working population. These were modeled with: t_* = time step at which the war shock occurs

$$\dot{k}(t = t_*) = -0.1k$$

$$\dot{P}(t = t_*) = -0.1k$$

As seen, the shock not only causes an immediate loss of capital, but also causes capital growth to lag behind for a significant period of time. However, in each case, the capital still approaches the same asymptotically stable equilibrium in the long term, despite the shock.



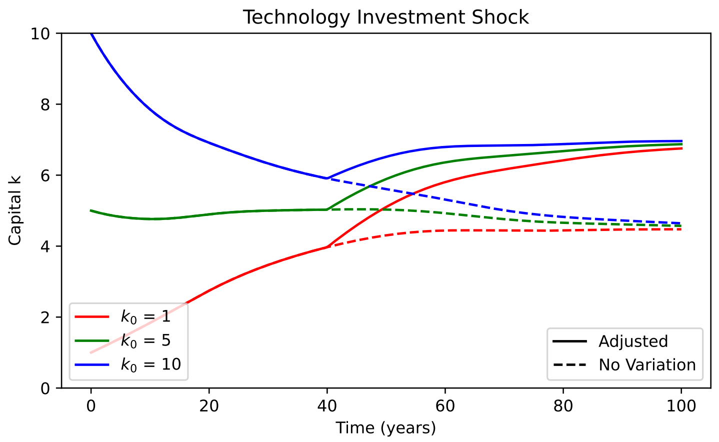
3.2. **Technology.** Technology is a measure of a country's ability to efficiently utilize capital. This could correspond to education, research and development, or efficient manufacturing technology. In this case, we modeled the shock of the government's large investment in or discovery of new technology, specifically an instantaneous 25% increase in the country's technological capabilities. The new technology function is given as follows:

$$A(t) = 1 \text{ if } t \leq 40$$
$$A(t) = 1.25 \text{ otherwise}$$

In this case, the country's capital growth rate increased significantly, ultimately trending towards a new, asymptotically stable equilibrium.

In the case of the war, the country experienced a momentary decrease in capital and capital growth, but trended toward the same long-term equilibrium. This is because the change in capital and population essentially provided a new set of initial conditions, but did not change any model parameters.

In contrast, the shock of the investment in technology created an entirely new equilibrium. $A(t)$ is a parameter of the income function, ultimately



acting as a model parameter. By increasing the technology, it created a new, higher equilibrium solution that the system tended to after the investment.

These behaviors are consistent with real-world economies. For example, despite the loss of capital and life in Germany, Japan, and other countries in WWII, the long term economic growth of the countries has been robust, today producing similar GDP per capita as the United States, a country that did not experience loss of life and capital to the same degree [3][4]. Over the last century, China has invested heavily in rapid industrialization, experiencing large GDP per capita growth as it has done so [5]. The SGM demonstrated itself to be robust and realistic in response to these shocks.

3.3. **Worker Productivity.**

The SGM can also be used to understand the effects of worker productivity on the GDP growth of a country. To do this, we introduced a variation to the SGM. This variation accounted for different levels of productivity depending on the age of the worker. The age groups were divided into 0-18 years, 18-40 years, 40-65 years, and 65-85 years. To account for different levels of productivity among these groups, we assigned an average productivity weight to each.

Worker productivity is reflected in the production function of the SGM, traditionally defined as: $y(t) = A(t)k(t)^{\alpha}$. We accounted for age-group productivity by introducing the vector $p \in \mathbb{R}^4$. $\vec{p}$ contains the average productivity weights for each age group, and $n(t)$ is the proportion of population in each age group at each time t given by the population SIR model. We adjusted the model by taking the standard inner product of p and n, normalizing this term with respect to n, and multiplying this term to our production function, as follows.

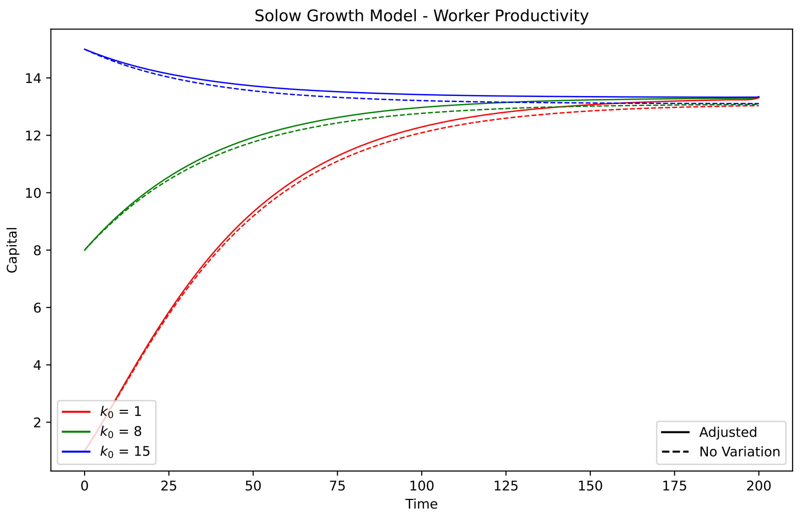
$$y(t) = A(t)k(t)^{\alpha}\frac{p \cdot \vec{n}}{\|\vec{n}\|_1}$$

Despite the population shifts, for this model, we assumed that the labor force growth rate is still constant, with a value of 0.01.

Identifying proper weights for each age group is important to create an accurate productivity model. The weights only matter relative to one another. We chose weights to reflect workers becoming increasingly productive until retirement, with the 41-65 age group given a weight of 1. For our worker productivity model, we chose the weights to be $p = [0.1, 0.8, 1.0, 0.3]$.

The standard SGM assumes that the entire population has equal productivity levels. Therefore, by averaging the weights of $\vec{p}$, we can depict the standard SGM and compare it to our variation. The standard model fits into the framework by using a constant $\overline{p}$ vector. In this case, $\overline{p} = [0.55, 0.55, 0.55, 0.55]$.

The graphs below show the difference between the typical SGM along with the worker productivity variation for a few different initial capital conditions.



Based on the graph above, we can see that adjusting the worker productivity led to a slightly higher steady-state solution than the original model, but the shapes are similar.

Since the weights were chosen somewhat arbitrarily, the exact relation between the adjusted model and the original model is not significant. However, the new steady-state solution does show that changes in relative productivity have a long-term impact on the GDP potential for an economy.

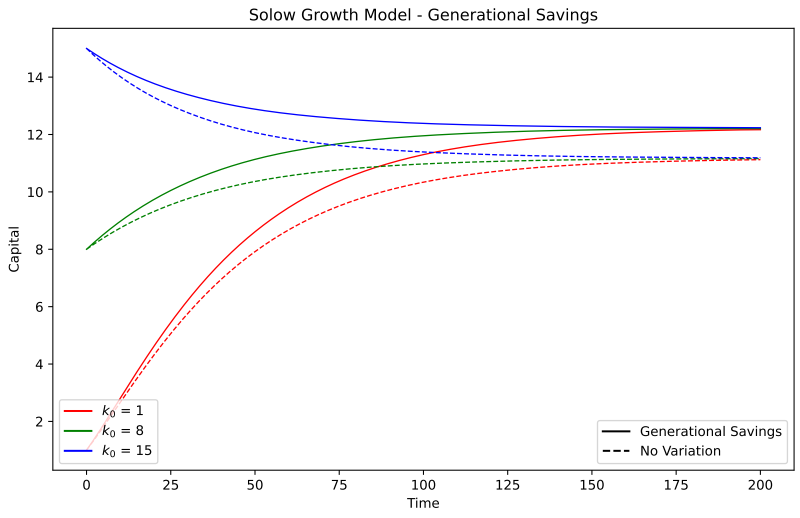
Both models have similar shapes and long-term behaviors, reflecting the robustness of the SGM in regards to worker productivity. By adding a reasonable assumption about workers, the SGM maintains its general form. Therefore, we conclude that the SGM's assumptions regarding worker productivity, while unreasonable, do not significantly affect the outcome.

3.4. **Generational Savings.** We also used the SGM to analyze the effect of different generational saving habits by relaxing the universal constant savings rate assumption. Similar to the worker productivity model, we partitioned the saving habits of each generation by assigning weights to each age group. The final two equations of the SGM are now given by:

$$\frac{\vec{s}\cdot\vec{n}}{\|\vec{n}\|_1}y(t) = i(t)$$

$$\dot{k} = \frac{\vec{s}\cdot\vec{n}}{\|\vec{n}\|_1}y(t) - (\delta + \Delta n)k(t)$$

The key difference in this model variation is that the constant s becomes a vector $\vec{s}$. We took the standard inner product of $\vec{s}$ and $\vec{n}$, then normalized by population. The labor force growth rate $\Delta\bar{n}$ is still assumed to be constant. We define the savings rate vector to be $\vec{s} = [0.2, 0.4, 0.4, 0.2]$. We selected weights such that individuals not generally in the labor force (ages 0-18 and 65+), save at 1/5th of the rate of those in the labor force (ages 19-65). The graph below compares this savings model to the standard model. Again, the standard model can be fit into this framework by averaging the weights and creating a constant $\bar{s}$ vector. In this case, $\bar{s} = [0.3, 0.3, 0.3, 0.3]$, which is the typical savings rate used.



Their overall behavior is similar, but generational savings do lead to an increase in capital growth and steady state. We conclude that the savings rate plays a major role in determining the generated capital: by weighting the savings vector toward individuals ages 19-65, the steady-state solution increases significantly. This model would indicate that if working individuals saved more money, then overall GDP would increase, but it should be noted that this model does not show how increased saving would affect firms' production, which also affects GDP. More research would be needed to fully understand the effect of personal savings on the overall economy.

Since the shape of the variation graph is similar to the original model, we can conclude that the SGM is robust to partitioning the savings rate.

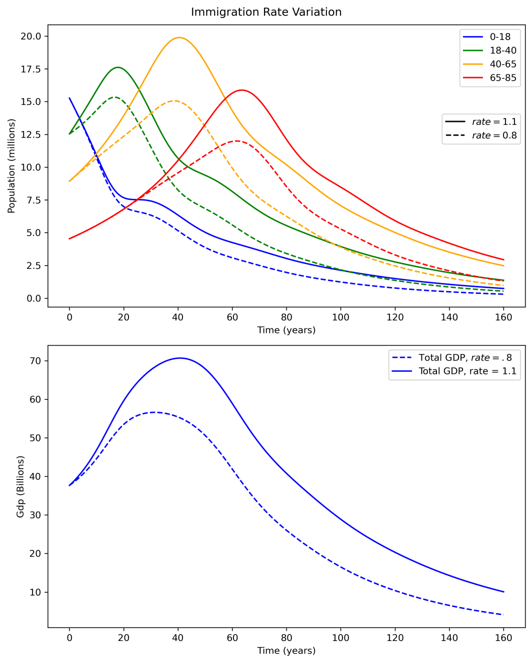
3.5. **Immigration.** An indispensable determinant of GDP growth on a global scale is immigration between countries. While international migrants constitute only 3.6% of the world's population, the World Immigration Report [6] found that some countries, for example Kuwait and Qatar, have populations comprised of more than 50% immigrants. Moreover, a study conducted by the Center for Immigration Studies underscores the substantial role of immigration as the primary driver of population growth in the United States [7]. In this section we investigate the effect of immigration on countries that are suffering from lower fertility rates and observe how this affects their GDP growth.

We modeled the impact of immigration patterns by modifying the SGM to include a new parameter, immigration_rate. When set to 1, this signifies a state of equilibrium with no migration flux. Values greater than 1 denote a net influx of individuals into the county, values less than 1 indicate a net outflow. For simplicity, we assumed that migrants fall strictly within the age range of 18 to 40, which is reflective of the age group of most immigrants.Their productivity levels are assumed to be consistent with the rest of the population.

We modeled both an economy with a net positive immigration flux(immigration_rate = 1.1) and one with a net negative immigration flux (immigration_rate = 0.8). We observed that a net positive immigration rate also leads to an increased number of births and an increase in overall population. While we expected that the total GDP per person would decrease in this scenario as it would be divided among more individuals, we observed no significant change in this quantity. A potential explanation for this could be that the number of immigrants isn't large enough to make a significant impact on earnings per person. Also, because we assumed everyone works with the same productivity, the GDP per person even increased over time, amidst fast population growth.

Conversely, under a net negative immigration flux, the GDP and population declined. Population aging and a lack of workforce replacement negatively impacted the country's economy. These findings suggest that immigration can have a positive impact on an economy, even when the number of immigrants is relatively small. However, it is important to note that the impact of immigration on GDP per person is complex and can depend on a variety of factors, including the skills and qualifications of the immigrants, the country's existing economic conditions, and the policies in place to support immigrants."

Notably, increased immigration impacted all age groups, despite the initial assumption that only individuals aged 18 to 40 would immigrate. This



modeling is particularly pertinent for nations struggling with the challenge of insufficient population growth, a circumstance exemplified by countries such as Japan[8] and Korea[9], where rapidly aging populations pose a threat to sustained economic growth. Similarly, certain European countries, like Germany and Italy, are experiencing demographic imbalances that jeopardize sustained GDP growth. This model underscores the interconnected nature of demographic shifts and holds significant relevance for guiding countries in shaping their immigration policies, particularly the role it plays in sustaining economic growth amidst declining fertility rates

4. Conclusion

Our modified Solow Growth Model, despite its simplicity, has proven itself to be a robust tool for examining different aspects of economic development's response to societal dynamics. A future investigation could seek to empirically measure the various model parameters, something that was outside of the scope of this project. Despite our use of arbitrary quantities, our model provided valuable qualitative insights into the economy's response to the tested factors. Moreover, with further examination and research, this model could serve as a practical guide for governments in formulating policies to better design welfare programs, incentivize beneficial worker and consumer habits, and shape immigration policies to effectively contribute to their country's economic advancement.

References

[1] Solow, Robert M. “A contribution to the theory of economic growth.” The Quarterly Journal of Economics, vol. 70, no. 1, 1956, p. 65, https://doi.org/10.2307/1884513. Accessed 21 Nov. 2023.

[2] World Bank, accessed 7 December 2023 https://data.worldbank.org/indicator/SP.DYN.TFRT.IN? locations=US-IN

[3] ”The Japanese Economic Miracle.” The Berkeley Economic Review, January 26, 2023. URL: econreview.berkeley.edu/the-japanese-economic-miracle/. Accessed 21 Nov. 2023.

[4] Henderson, David R. “German Economic Miracle.” Econlib, 9 Nov. 2021, www.econlib.org/library/Enc/GermanEconomicMiracle.html. Accessed 21 Nov. 2023.

[5] Zuliu Hu and Mohsin S. Khan. (June 1, 1997). “Why Is China Growing So Fast?”. International Monetary Fund. Retrieved from https://www.imf.org/external/pubs/ft/issues8/index.htm#Author

[6] “World Migration Report 2022.” World Migration Report, UN Migration, worldmigrationreport.iom.int/wmr-2022-interactive/. Accessed 21 Nov. 2023.

[7] Camarota, Steven A, and Karen Zeigler. “Estimating the Impact of Immigration on U.S. Population Growth.” CIS.Org, Center for Immigration Studies, 27 Mar. 2023, cis.org/Report/Estimating-Impact-Immigration-US-Population-Growth. Accessed 21 Nov. 2023.

[8] Jack, D. (2016). “The Issue of Japan’s Aging Population.” University of Chicago Law School, retrieved from chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=1034& context=international_immersion_program_papers

[9] Poston Jr, D. L. (June 27, 2023). “South Korea has the lowest fertility rate in the world – and that doesn’t bode well for its economy.” Texas A&M University. Retrieved from theconversation.com/south-korea-has-the-lowest-fertility-rate-in-the-world-and-that-doesnt-bode-well-for-its-economy-207107

Volume4ProjectCode

December 7, 2023

```
[ ]: import numpy as np
     from matplotlib import pyplot as plt
     from scipy.integrate import solve_ivp
     from matplotlib.lines import Line2D
     import scipy.linalg as la
```

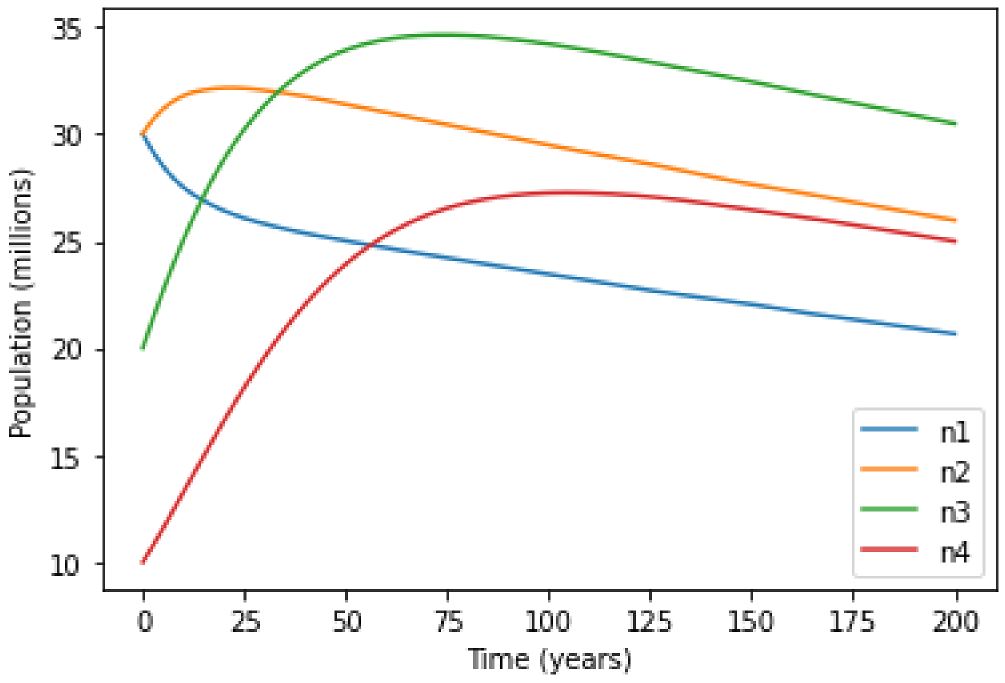
1 Population Augmented-SIR Model

```
[ ]: def get_deltas(age_brackets=[18, 40, 65, 85]):
         """
         Helper function that calculates and returns a list of deltas based on␣
      ↪birthrate and age brackets.

         Parameters:
         - age_brackets (list): A list of age brackets defining the population␣
      ↪segments.

         Returns:
         - list: A list of deltas calculated based on the given birthrate and age␣
      ↪brackets.
         """

         # Insert 0 at the beginning of age_brackets to simplify calculations
         M = [0] + age_brackets

         # Calculate deltas based on age brackets
         deltas = [1 / (M[i] - M[i - 1]) for i in range(1, len(M))]

         # This is the births delta
         deltas = [.5/(M[2]-M[1])]+deltas

         # Return the modified delta list
         return deltas
```

```
[ ]: def population_SIR(deltas,fertility_rate = 2):
         """
```

```
    Creates a function representing the population dynamics in an SIR␣
↪(Susceptible-Infectious-Recovered) model.

    Parameters:
    - deltas (list or array): A list or array containing the transition rates␣
↪between different population compartments.
    - birthrate (int or float or function) The birthrate for the country. Can␣
↪be constant or a function of time.

    Returns:
    - function: A function representing the population dynamics in the SIR␣
↪model.
    """

    # Convert deltas to a NumPy array for numerical operations
    d = np.array(deltas)

    def ode(t, n):
        """
        Function representing the rate of change of each population compartment␣
↪in the SIR model.

        Parameters:
        - t (float): Time parameter (not used in the function, but required for␣
↪integration).
        - n (array): Array representing the current state of the population␣
↪compartments.

        Returns:
        - array: Array representing the rate of change of each population␣
↪compartment.
        """
        DN = np.zeros_like(n)

        # Calculate the rate of change for each compartment based on the SIR␣
↪model equations
        if callable(fertility_rate):
            DN[0] = fertility_rate(t)*d[0] * n[1] - d[1] * n[0]
        else:
            DN[0] = fertility_rate*d[0] * n[1] - d[1] * n[0]
        DN[1:] = -d[2:] * n[1:] + d[1:-1] * n[:-1]

        return DN

    # Return the function representing the population dynamics
    return ode
```

```
[ ]: deltas = get_deltas()
     ode = population_SIR(deltas,1.9)

     ts = np.linspace(0,200,512)
     x0 = np.array([30,30,20,10])

     sol =solve_ivp(ode,(0,200),x0,t_eval=ts)

     plt.plot(sol.t,sol.y.T,label = ["n1","n2","n3","n4"])
     plt.legend()
     plt.xlabel("Time (years)")
     plt.ylabel("Population (millions)")
     plt.show()
```



```
[ ]: %store get_deltas
     %store population_SIR
     %store sol
```

```
Warning:get_deltas is <function get_deltas at 0x7f480fd13e20>
Proper storage of interactively declared classes (or instances
of those classes) is not possible! Only instances
of classes in real modules on file system can be %store'd.
```

```
Warning:population_SIR is <function population_SIR at 0x7f4807a4d090>
Proper storage of interactively declared classes (or instances
of those classes) is not possible! Only instances
of classes in real modules on file system can be %store'd.

Stored 'sol' (OdeResult)
```

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import solve_ivp

class Solow_Model_Parameters():
    """This class holds the parameters for the Solow Growth Model."""

    def __init__(self, A=1, alpha=0.5, delta=0.08, s=0.3, weights=lambda x: (x␣
↪>= 18) * (x <= 65)):
        """
        Initializes the parameters for the Solow Growth Model.

        Parameters:
        - A: Total factor productivity (Number or function with respect to time)
        - alpha: Output elasticity of capital
        - delta: Depreciation rate
        - s: Savings rate  (Number or function with respect to time)
        - weights: Function defining the age distribution weights
        """
        self.A = A
        self.alpha = alpha
        self.delta = delta
        self.s = s
        self.weights = weights

    def y(self, k, t, n=None):
        """
        Computes output (Y) given the capital (K), time (t), and labor (N).

        Parameters:
        - k: Capital
        - t: Time
        - n: Labor (optional). Proportion of the total population that is␣
↪considered the labor force OR can be used to incorporate probability levels

        Returns:
        - Output (Y)
        """
        if callable(self.A):
            A = self.A(t)
        else:
            A = self.A
        if n is None:
            return A * k ** self.alpha
        else:
            return A * k ** self.alpha * n

    def kprime(self, k, dpop, pop, start_working, retire, t):
        """
        Computes the change in capital (K') given the capital (K), population␣
 ↪change (dpop),
        total population (pop), start_working age, retire age, and time (t).

        Parameters:
        - k: Capital
        - dpop: Change in population
        - pop: Total population
        - start_working: Starting age of the workforce
        - retire: Retirement age
        - t: Time

        Returns:
        - Change in capital (K')
        """
        if callable(self.s):
            s = self.s(t)
        else:
            s = self.s
        n = (dpop[start_working] - dpop[retire + 1]) / np.sum(pop[start_working:
 ↪retire + 1])
        return s * self.y(k, t,np.sum(pop[start_working:retire+1],axis =0)/np.
 ↪sum(pop,axis = 0)) - (self.delta + n) * k

class Solution():
    """This class holds results from the Ordinary Differential Equation (ODE)␣
 ↪solution."""

    def __init__(self, t, population, capital=None, y=None):
        """
        Initializes the solution results for the ODE.

        Parameters:
        - t: Time
        - population: Total population
        - capital: Capital (optional)
        - y: Output (Y) (optional)
        """
        self.t = t
        self.population = population
        self.capital = capital
        self.y = y
```

[ ]:
```
import numpy as np
from scipy.integrate import solve_ivp

class Population_Solow_Model():
    def __init__(self, life_expectancy=85, fertility_rate=2.0,␣
 ↪fertility_starts=18, fertility_ends=40,␣
 ↪solow_growth_parameters=None,shock=False, imigration_rate=1):
        """
        Initializes the Population Solow Model with parameters.

        Parameters:
        - life_expectancy: The average life expectancy in the model. Must be an␣
 ↪integer and at least 40. Default is 85.
        - fertility_rate: A constant fertility rate or a function of time.␣
 ↪Default is 2.0.
        - fertility_starts: The age at which fertility begins. Default is 18.
        - fertility_ends: The age at which fertility ends. Default is 40.
        - solow_growth_parameters: Parameters for the Solow Growth Model.␣
 ↪Default is None.
        """
        self.fertility_rate = fertility_rate
        self.imigration_rate = imigration_rate      # Add another parameter to␣
 ↪measure imigration

        # Validate input types
        if not isinstance(life_expectancy, int):
            raise TypeError("'life_expectancy' must be an integer in this model.
 ↪")
        self.life_expectancy = life_expectancy

        if not isinstance(fertility_starts, int):
            raise TypeError("'fertility_starts' must be an integer in this␣
 ↪model.")
        self.fertility_starts = fertility_starts

        if not isinstance(fertility_ends, int):
            raise TypeError("'fertility_ends' must be an integer in this model.
 ↪")
        self.fertility_ends = fertility_ends
        self.ode = None
        self.labels = None
        self.start_working = None
        self.retire = None
        self.SGP = solow_growth_parameters
        self.include_SGP = False
        self.shock = shock
        return

    def prep_model(self, start_working=18, retire=65):
        """
        Prepares the population model based on the SIR framework with granular␣
↪fertility.

        Parameters:
        - start_working: The age at which individuals start working. Default is␣
↪18.
        - retire: The retirement age. Default is 65.

        Returns:
        - self: The Population Solow Model instance with prepared settings.

        Raises:
        - TypeError: If life_expectancy is not an integer.
        - ValueError: If life_expectancy is less than 40.
        """
        # Validate life expectancy
        if not isinstance(self.life_expectancy, int):
            raise TypeError("Life expectancy must be an integer in this model.")
        elif self.life_expectancy < 40:
            raise ValueError("This model requires life_expectancy to be at␣
↪least 40.")

        self.include_SGP = self.SGP is not None

        # Define the ordinary differential equation (ODE)
        def ode(t, x):
            """
            Ordinary differential equation representing the SIR population␣
↪dynamics with granular fertility.

            Parameters:
            - t: Time variable.
            - x: Array representing the state variables, where x[i] represents␣
↪the number of people at age i.

            Returns:
```

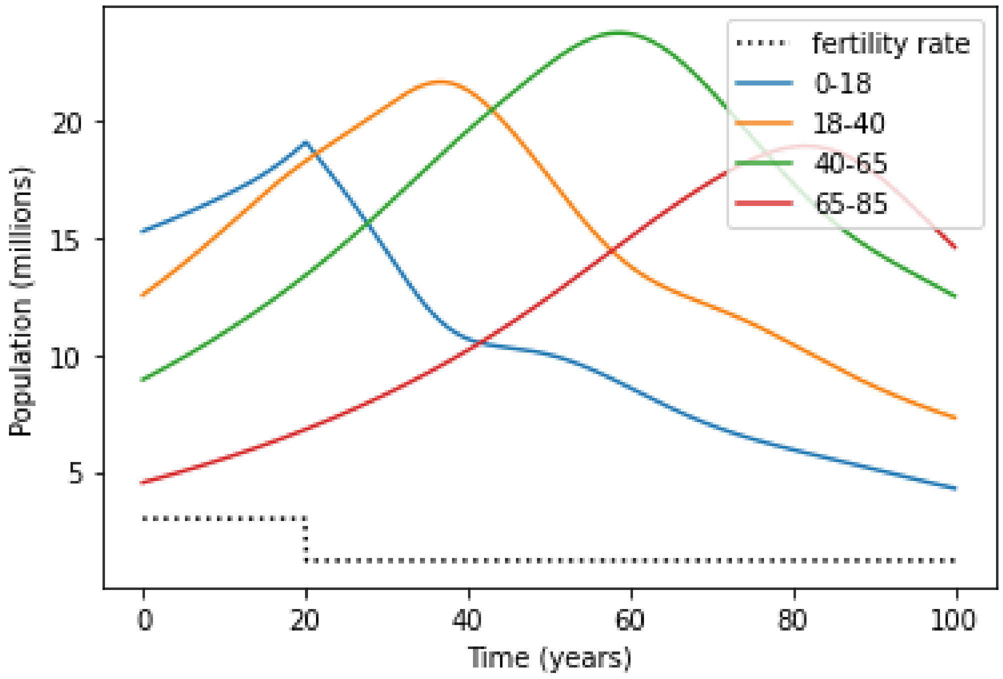
```
            - dx: Array representing the rates of change of the state variables.
            """
            dx = np.zeros_like(x)

            # Calculate the births based on the fertility rate
            if callable(self.fertility_rate):
                dx[0] = 0.5 * self.fertility_rate(t) * np.mean(x[self.
↪fertility_starts:self.fertility_ends + 1]) - x[0]
            else:
                dx[0] = 0.5 * self.fertility_rate * np.mean(x[self.
↪fertility_starts:self.fertility_ends + 1]) - x[0]

            # Calculate the change in other age categories
            dx[1:] = x[:-1] - x[1:]
            dx[18:40] += (self.imigration_rate-1)*x[18:40]/(40-18)

            # Add the capital allocation variables if necessary
            if self.include_SGP:
                dx[-1] = self.SGP.kprime(x[-1], dx[:-1], x[:-1], start_working,␣
↪retire, t)

            #If we're doing a war shock, include this
            if(self.shock):
                if 19.5<= t and t <=20.5:
                    dx[18:40] = -.1*x[18:40]
                    dx[-1] = -.1*x[-1]

            return dx

        # Set attributes for the model
        self.ode = ode
        self.labels = [f"{i}-{i+1}" for i in range(self.life_expectancy)]
        self.start_working = start_working
        self.retire = retire
        return self

    def solve(self, t_points, starting_population, starting_capital=None):
        """
        Solves the ODE for population dynamics given initial conditions.

        Parameters:
        - t_points: Time points to solve the ODE.
        - starting_population: Initial population distribution across age␣
↪categories.
        - starting_capital: Initial capital (if Solow Growth Parameters are␣
↪included).
```

```
        Returns:
        - Solution: Instance of the Solution class containing the ODE results.

        Raises:
        - AssertionError: If Population_Solow_Model().ode is not defined.
        """
        if self.ode is None:
            raise AssertionError("'Population_Solow_Model().ode' has not been␣
↪defined. Run 'Population_Solow_Model().create_model' first.")

        # Prepare initial state
        if not self.include_SGP:
            x = starting_population
        else:
            x = np.concatenate([starting_population, np.
↪array([starting_capital])])

        t_span = (np.min(t_points), np.max(t_points))

        # Solve the ODE
        sol = solve_ivp(self.ode, t_span, x, t_eval=t_points)

        if self.SGP is None:
            return Solution(sol.t, sol.y)
        else:
            # Calculate labor and return Solution instance
            n = np.sum(sol.y[:-1] * np.reshape(self.SGP.weights(np.arange(self.
↪life_expectancy)), (-1, 1)), axis=0) / np.sum(sol.y[:-1], axis=0)
            return Solution(sol.t, sol.y[:-1], sol.y[-1], self.SGP.y(sol.y[-1],␣
↪ts, n))
```

[ ]:
```
def consolidate_age_groups(x,age_brackets = [18,40,65],return_labels = False):
    age_brackets.sort()
    M = [0]+age_brackets + [len(x)]
    ns = np.array([np.sum(x[M[i]:M[i+1]],axis = 0)for i in range(len(M)-1)])
    if return_labels:
        labels = [f"{M[i]}-{M[i+1]}" for i in range(len(M)-1)]
        return ns,labels
    else:
        return ns
```

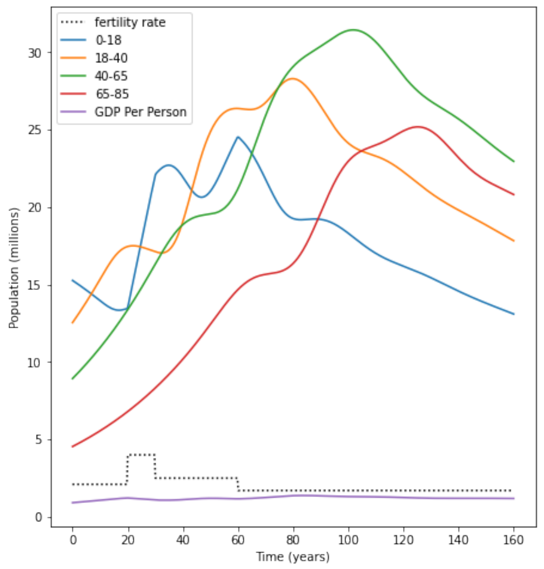
[ ]:
```
life_expect = 85
def fertility_rate(t):
    #Either a constant, or a function with respect to time
    return 1.2 + 1.8*(t<=20)
# Here we initialize the ode function to solve
```

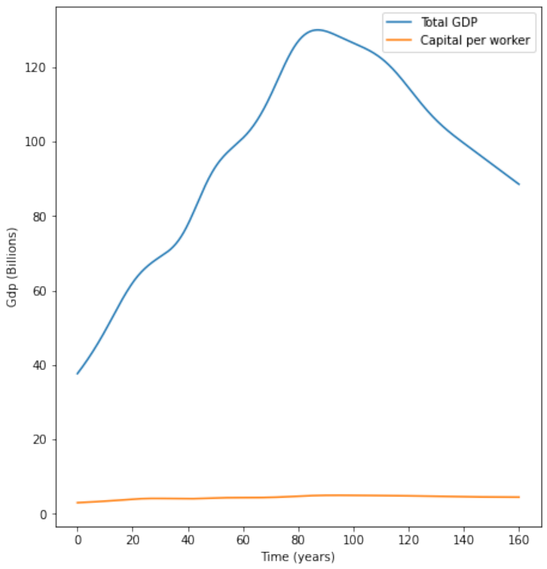
```
population_model =␣
 ↪Population_Solow_Model(fertility_rate=fertility_rate,life_expectancy=life_expect)

ts = np.linspace(0,100,512)
# Here we set the initial population
x0 = np.exp(-.02*np.arange(life_expect))

population_model.prep_model()
sol = population_model.solve(ts,x0,2)



# Here consolidate the ages into age brackets, to make the data more visible
# Grouping the age brackets will also make it easier for our model
values,labels = consolidate_age_groups(sol.population,return_labels=True)

# Here we plot the results
plt.plot(sol.t,fertility_rate(sol.t),":k",label = "fertility rate")
plt.plot(sol.t,values.T,label = labels)
plt.legend()
plt.xlabel("Time (years)")
plt.ylabel("Population (millions)")
plt.show()
```



```
[ ]: life_expect = 85
     def fertility_rate(t):
         #Either a constant, or a function with respect to time
         return 2.1 + 1.9*(t>=20)-1.5*(t>=30) - .8*(t>=60)

     solow_parameters = Solow_Model_Parameters()
     # Here we initialize the ode function to solve
     population_model =␣
       ↪Population_Solow_Model(fertility_rate=fertility_rate,life_expectancy=life_expect,solow_grow

     ts = np.linspace(0,160,512)
     # Here we set the initial population
     x0 = np.exp(-.02*np.arange(life_expect))

     population_model.prep_model()
     sol = population_model.solve(ts,x0,3)
     # Here consolidate the ages into age brackets, to make the data more visible
     # Grouping the age brackets will also make it easier for our model
     values,labels = consolidate_age_groups(sol.population,return_labels=True)

     # Here we plot the results
     print("With capital this time as well.")
     plt.figure(figsize=(16,8))
     plt.subplot(121)
     plt.plot(sol.t,fertility_rate(sol.t),":k",label = "fertility rate")
     plt.plot(sol.t,values.T,label = labels)
     plt.plot(sol.t,sol.y,label = "GDP Per Person")
     plt.legend()
     plt.xlabel("Time (years)")
     plt.ylabel("Population (millions)")

     plt.subplot(122)
     plt.xlabel("Time (years)")
     plt.ylabel("Gdp (Billions)")

     plt.plot(sol.t,sol.y*np.sum(sol.population,axis = 0),label = "Total GDP")
     plt.plot(sol.t, sol.capital,label = "Capital per worker")
     plt.legend()
     plt.show()
```

```
With capital this time as well.
```



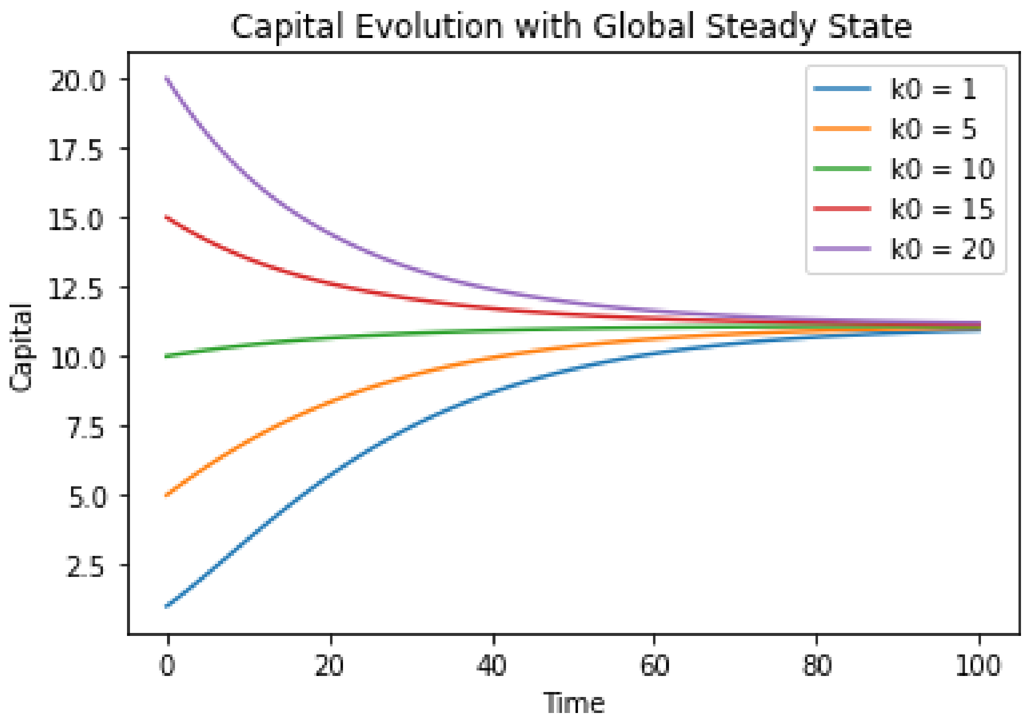


2 Baseline Solow Growth Model

[ ]:
```
#Define a function to return the income per person in the economy.
def y(k, alpha=0.5, A=1):
    return A*k**alpha

#The captital evolution equation
def kprime(k, alpha=0.5, s=0.3, delta=0.08, n=0.01, A=1):
    return s*y(k, alpha, A) - (delta + n)*k
```

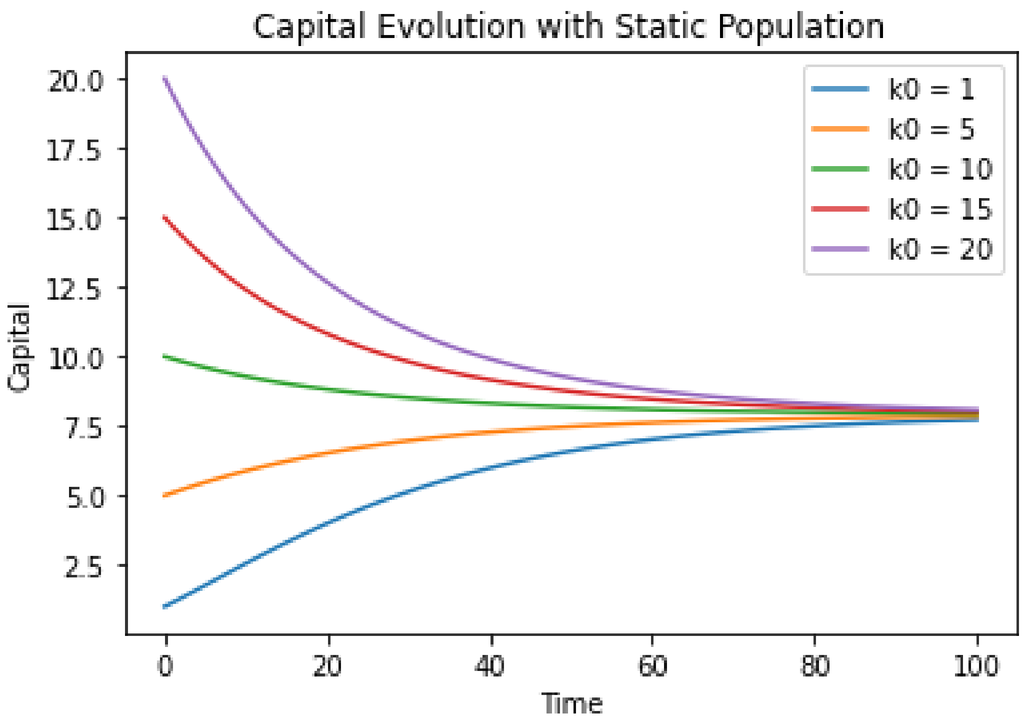
[ ]:
```
#A very simple model where capital trends towards a global steady state
def ksolve(t, k, alpha=0.5, s=0.3, delta=0.08, n=0.01, A=1):
    return kprime(k, alpha=0.5, s=0.3, delta=0.08, n=0.01, A=1)

#Time domain
t_span = (0,100)
T = np.linspace(0,100,101)

#Initial conditions
k0 = [1]
k1 = [5]
k2 = [10]
k3 = [15]
k4 = [20]

#Solve with different initial conditions
sol0 = solve_ivp(ksolve, t_span, k0, t_eval=T)
```

```
sol1 = solve_ivp(ksolve, t_span, k1, t_eval=T)
sol2 = solve_ivp(ksolve, t_span, k2, t_eval=T)
sol3 = solve_ivp(ksolve, t_span, k3, t_eval=T)
sol4 = solve_ivp(ksolve, t_span, k4, t_eval=T)


#Make the plots
f = plt.figure()
plt.plot(T, sol0.y.reshape(-1), label='k0 = 1')
plt.plot(T, sol1.y.reshape(-1), label='k0 = 5')
plt.plot(T, sol2.y.reshape(-1), label='k0 = 10')
plt.plot(T, sol3.y.reshape(-1), label='k0 = 15')
plt.plot(T, sol4.y.reshape(-1), label='k0 = 20')

plt.xlabel('Time')
plt.ylabel('Capital')
plt.legend()
plt.title("Capital Evolution with Global Steady State")
plt.show()
f.savefig("global_steady_state.pdf", bbox_inches='tight')
```



```
[ ]: def y_pop(k, N = np.array([0.2, 0.3, 0.3, 0.2]), C=np.array([0,1,1.5, 0]),␣
     ↪alpha=0.5, A=1):
```

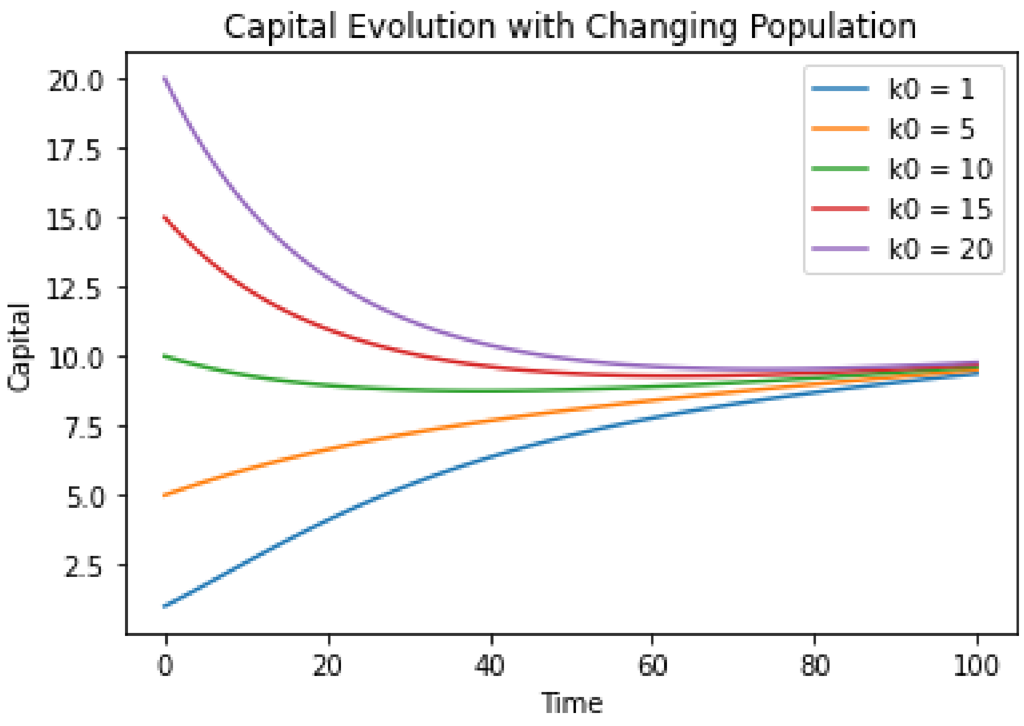
```
    #N is a vector with the breakdown of population by age
    #C is a vector with the contribution of each age demographic

    return np.inner(N, C)*y(k, alpha=alpha, A=A)

#Now we solve the thing with these slight modifications
def kprime_pop(k,  N = np.array([0.2, 0.3, 0.3, 0.2]), C=np.array([0,1,1.5,␣
 ↪0]), alpha=0.5, s=0.3, delta=0.08, n=0.0, A=1):
    return s*y_pop(k, N, C, alpha, A) - (delta + n)*k

def ksolve_pop(t, k, N = np.array([0.2, 0.3, 0.3, 0.2]), C=np.array([0,1,1.5,␣
 ↪0]), alpha=0.5, s=0.3, delta=0.08, n=0.0, A=1):
    return kprime_pop(k, N, C, alpha, s, delta, n, A)

#Time domain
t_span = (0,100)
T = np.linspace(0,100,101)

#Initial conditions
k0 = [1]
k1 = [5]
k2 = [10]
k3 = [15]
k4 = [20]

#Solve with different initial conditions
sol0 = solve_ivp(ksolve_pop, t_span, k0, t_eval=T)
sol1 = solve_ivp(ksolve_pop, t_span, k1, t_eval=T)
sol2 = solve_ivp(ksolve_pop, t_span, k2, t_eval=T)
sol3 = solve_ivp(ksolve_pop, t_span, k3, t_eval=T)
sol4 = solve_ivp(ksolve_pop, t_span, k4, t_eval=T)


#Make the plots
plt.plot(T, sol0.y.reshape(-1), label='k0 = 1')
plt.plot(T, sol1.y.reshape(-1), label='k0 = 5')
plt.plot(T, sol2.y.reshape(-1), label='k0 = 10')
plt.plot(T, sol3.y.reshape(-1), label='k0 = 15')
plt.plot(T, sol4.y.reshape(-1), label='k0 = 20')

plt.xlabel('Time')
plt.ylabel('Capital')
plt.legend()
plt.title("Capital Evolution with Static Population")
plt.show()

print(sol0.y[:, -1])
```



```
[7.72424666]
```

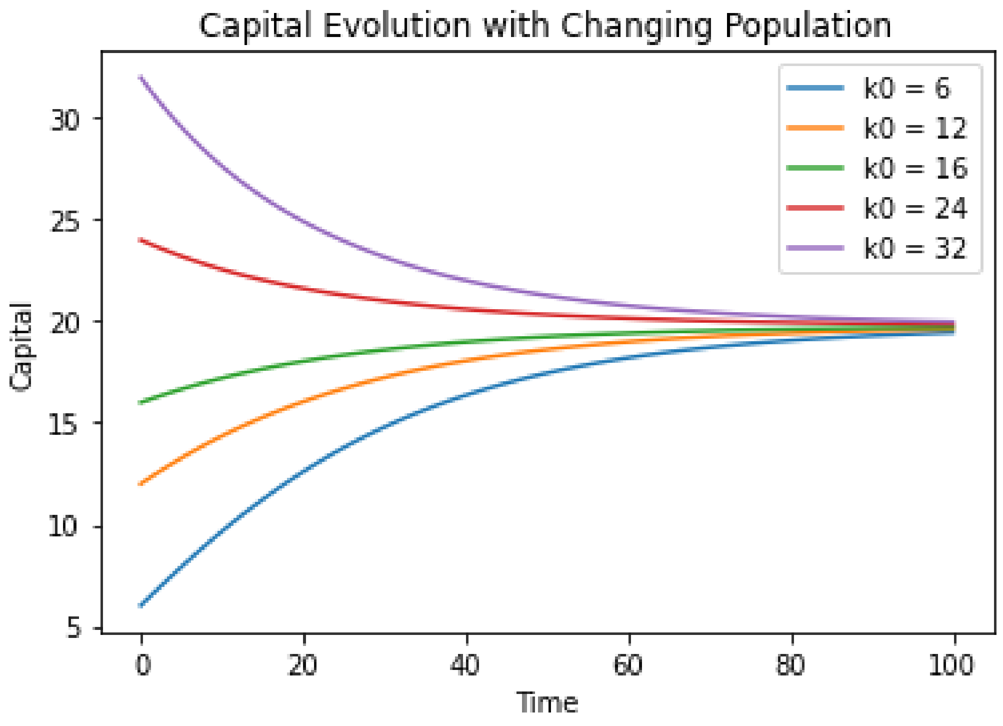
[ ]:
```
def age(t):
    #A slight aging of the population
    return np.array([0.2 - t/500, 0.3-t/500, 0.3+t/500, 0.2+t/500])

def ksolve_pop2(t, k, N = age, C=np.array([0,1,1.5, 0]), alpha=0.5, s=0.3,␣
 ↪delta=0.08, n=0.00, A=1):
    N_t = N(t)
    return kprime_pop(k, N_t, C, alpha, s, delta, n, A)

#Time domain
t_span = (0,100)
T = np.linspace(0,100,101)

#Initial conditions
k0 = [1]
k1 = [5]
k2 = [10]
k3 = [15]
k4 = [20]

#Solve with different initial conditions
sol0 = solve_ivp(ksolve_pop2, t_span, k0, t_eval=T)
```

```
sol1 = solve_ivp(ksolve_pop2, t_span, k1, t_eval=T)
sol2 = solve_ivp(ksolve_pop2, t_span, k2, t_eval=T)
sol3 = solve_ivp(ksolve_pop2, t_span, k3, t_eval=T)
sol4 = solve_ivp(ksolve_pop2, t_span, k4, t_eval=T)


#Make the plots
plt.plot(T, sol0.y.reshape(-1), label='k0 = 1')
plt.plot(T, sol1.y.reshape(-1), label='k0 = 5')
plt.plot(T, sol2.y.reshape(-1), label='k0 = 10')
plt.plot(T, sol3.y.reshape(-1), label='k0 = 15')
plt.plot(T, sol4.y.reshape(-1), label='k0 = 20')

plt.xlabel('Time')
plt.ylabel('Capital')
plt.legend()
plt.title("Capital Evolution with Changing Population")
plt.show()

print(sol0.y[:, -1])
```



```
[9.37727137]
```

```
[ ]: def expenditure(N = np.array([.2*10e8, .3*10e8, .3*10e8, .2*10e8]), S=np.
     ↪array([0, 0, 0, 35*10e3])):
         #N is a vector of the number of people in each age bracket
         #S is a vector of payment rates the government makes to those people
         return np.inner(N, S)

     def y_pop(k, N, C, alpha=0.2, A=1):
         #N is a vector with the breakdown of population by age
         #C is a vector with the contribution of each age demographic
         return np.inner(N, C) * k**alpha * A

     def kprime_pop2(k,  N, C, alpha=0.5, s=0.3, delta=0.08, n=0.0, A=1):
         return s*(y_pop(k, N, C, alpha, A)  - expenditure(N)) / np.sum(N) - (delta␣
     ↪+ n)*k

     age = np.array([.2*10e8, .3*10e8, .3*10e8, .2*10e8])
     def ksolve_pop3(t, k, N = age, C=np.array([0,50000,75000, 0]), alpha=0.5, s=0.
     ↪3, delta=0.08, n=0.00, A=1):
         N_t = N
         return kprime_pop2(k, N_t, C, alpha, s, delta, n, A)

     #Time domain
     t_span = (0,100)
     T = np.linspace(0,100,101)

     #Initial conditions
     k0 = [6*10e8]
     k1 = [12*10e8]
     k2 = [16*10e8]
     k3 = [24*10e8]
     k4 = [32*10e8]

     age = np.array([.2*10e8, .3*10e8, .3*10e8, .2*10e8])

     #Solve with different initial conditions
     sol0 = solve_ivp(ksolve_pop3, t_span, k0, t_eval=T)
     sol1 = solve_ivp(ksolve_pop3, t_span, k1, t_eval=T)
     sol2 = solve_ivp(ksolve_pop3, t_span, k2, t_eval=T)
     sol3 = solve_ivp(ksolve_pop3, t_span, k3, t_eval=T)
     sol4 = solve_ivp(ksolve_pop3, t_span, k4, t_eval=T)

     #Make the plots
     plt.plot(T, sol0.y.reshape(-1)/10e8, label='k0 = 6')
     plt.plot(T, sol1.y.reshape(-1)/10e8, label='k0 = 12')
     plt.plot(T, sol2.y.reshape(-1)/10e8, label='k0 = 16')
     plt.plot(T, sol3.y.reshape(-1)/10e8, label='k0 = 24')
     plt.plot(T, sol4.y.reshape(-1)/10e8, label='k0 = 32')
     plt.xlabel('Time')
     plt.ylabel('Capital')
     plt.legend()
     plt.title("Capital Evolution with Changing Population")
     plt.show()

     print(sol0.y[:, -1])
```



```
[1.94504596e+10]
```

3 Shocks

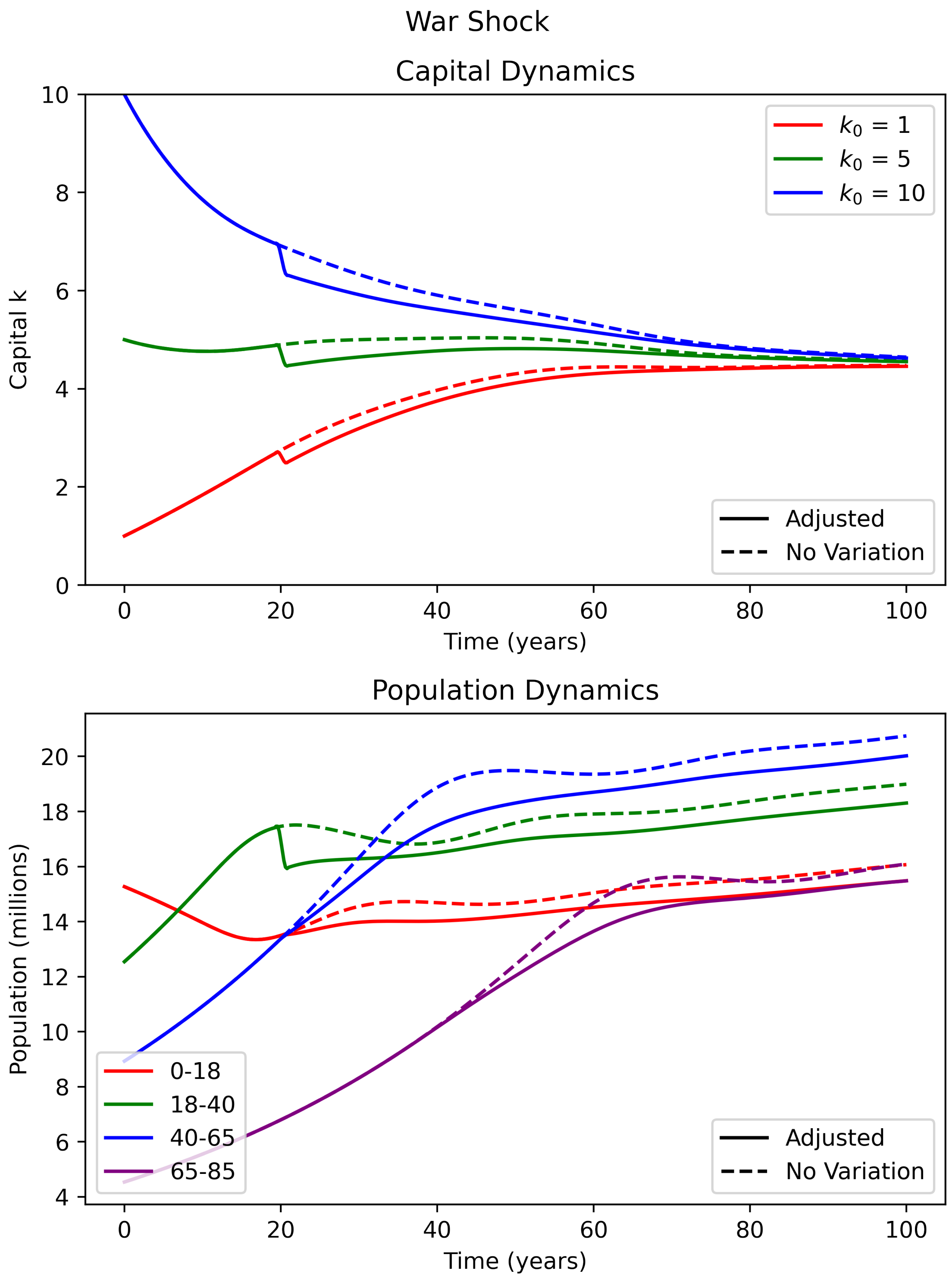
```
[ ]: #Define several k initial conditions, plotting conditions
     IC = [1,5,10]
     count = 0
     fig1 = plt.figure(figsize=(6,8),dpi=500)
     plt.subplot(211)
     label_list = ["Adjusted", "No Variation"]
     colors = ["red", "green", "blue"]#,"orange","purple"]
     line_style = ["--", "-"]
     color_legend = []
     style_legend = []
```

```
#Define the model without shocks
solow_parameters = Solow_Model_Parameters()
# Here we initialize the ode function to solve
shock_model = Population_Solow_Model(fertility_rate=2.
 ↪1,solow_growth_parameters=solow_parameters,shock=True)
prior_model = Population_Solow_Model(fertility_rate=2.
 ↪1,solow_growth_parameters=solow_parameters)
shock_model.prep_model()
prior_model.prep_model()


#Iterate through initial conditions, plot both the shock and the growth without␣
 ↪shock
for k0 in IC:
    # Here we initialize the ode function to solve for the shock situation
    # ode_shock = population_with_captial_growth_shocks(fertility_rate=2.1,␣
 ↪life_expectancy=85, fertility_starts=18, fertility_ends=40,A=1,alpha=.
 ↪5,delta=0.08,s=.3)
    # ode_normal = population_with_captial_growth(fertility_rate=2.1,␣
 ↪life_expectancy=85, fertility_starts=18, fertility_ends=40,A=1,alpha=.
 ↪5,delta=0.08,s=.3)
    ts = np.linspace(0,100,501)
    # Here we set the initial conditions, solve the ODE
    pop_0 = np.exp(-.02*np.arange(life_expect))

    sol_shock = shock_model.solve(ts,pop_0,k0)
    sol_normal = prior_model.solve(ts,pop_0,k0)

    # Here consolidate the ages into age brackets, to make the data more visible
    # Grouping the age brackets will also make it easier for our model
    pop_values_shock,pop_labels = consolidate_age_groups(sol_shock.
 ↪population,return_labels=True)
    pop_values_normal,pop_labels = consolidate_age_groups(sol_normal.
 ↪population,return_labels=True)

    #Plot capital
    color = colors[count]
    plt.plot(sol_shock.t,sol_shock.capital,'-',color = color)
    plt.plot(sol_normal.t,sol_normal.capital,'--',color = color)

    #Add color line to legend
    colorLine = Line2D([0,1],[0,1], linestyle='-', color=color)
    color_legend.append(colorLine)
    count = count + 1
```

```
#Make 2 legends, set other plot information
styleLine = Line2D([0,1],[0,1], linestyle='-', color="black")
style_legend.append(styleLine)
styleLine = Line2D([0,1],[0,1], linestyle='--', color="black")
style_legend.append(styleLine)
legend1 = plt.legend(color_legend, [r"$k_0$" + " = " + str(k0) for k0 in IC],␣
 ↪loc=1)
plt.legend(style_legend, label_list, loc=4)
plt.gca().add_artist(legend1)

plt.xlabel("Time (years)")
plt.ylabel("Capital k")
plt.suptitle(f"War Shock")
plt.title('Capital Dynamics')
plt.ylim([0,10])
plt.tight_layout()


# plt.plot(sol.t,sol.y*np.sum(sol.population,axis = 0),label = "Total GDP")
# plt.plot(sol.t, sol.capital,label = "Capital per worker")

# Here we plot the population growth
label_list = ["Adjusted", "No Variation"]
colors = ["red", "green", "blue","purple"]
color_legend = []
style_legend = []
plt.subplot(212)
for i in range(pop_values_shock.T.shape[1]):
    color = colors[i]
    to_plot_shock = pop_values_shock.T[:,i]
    to_plot_normal = pop_values_normal.T[:,i]
    plt.plot(sol_shock.t,to_plot_shock,'-',color = color)
    plt.plot(sol_normal.t,to_plot_normal,'--',color = color)
    #Add color line to legend
    colorLine = Line2D([0,1],[0,1], linestyle='-', color=color)
    color_legend.append(colorLine)
#Make 2 legends, set other plot information
styleLine = Line2D([0,1],[0,1], linestyle='-', color="black")
style_legend.append(styleLine)
styleLine = Line2D([0,1],[0,1], linestyle='--', color="black")
style_legend.append(styleLine)
legend1 = plt.legend(color_legend, pop_labels, loc=3)
plt.legend(style_legend, label_list, loc=4)
plt.gca().add_artist(legend1)

plt.xlabel("Time (years)")
plt.ylabel("Population (millions)")
```

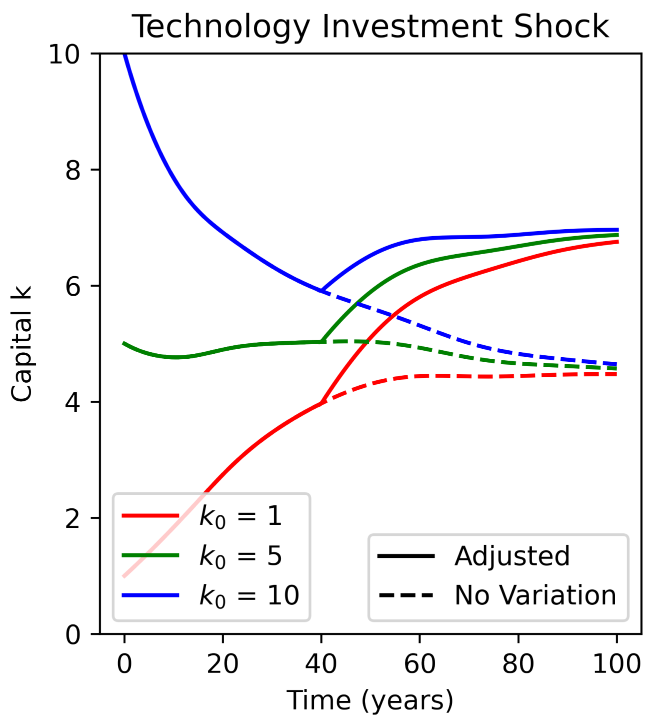
```python
plt.title("Population Dynamics")
plt.tight_layout()
plt.show()
fig1.savefig('Saved Figures/War Shock.pdf',bbox_inches = 'tight')
```

```
/tmp/ipykernel_14754/1748665256.py:44: RuntimeWarning: invalid value encountered
in double_scalars
  return A * k ** self.alpha * n
```


War Shock
Capital Dynamics
Capital k
Time (years)
$k_0 = 1$
$k_0 = 5$
$k_0 = 10$
Adjusted
No Variation
Population Dynamics
Population (millions)
Time (years)
0-18
18-40
40-65
65-85
Adjusted
No Variation

3.0.1 Technology Investment Shock

[ ]:
```
#Define an investment shock function
def A_investment(t):
    if 40 <= t:
        return 1.25
    else:
        return 1
A_investment = np.vectorize(A_investment)
#Define Initial conditions
IC = [1,5,10]
ts = np.linspace(0,150,501)

#Define plotting parameters
fig2 = plt.figure(figsize=(8,4),dpi=500)
plt.subplot(121)
label_list = ["Adjusted", "No Variation"]
colors = ["red", "green", "blue"]#,"orange","purple"]
line_style = ["--", "-"]
color_legend = []
style_legend = []

#Define the model
#Define the model without shocks
solow_parameters_tech = Solow_Model_Parameters(A=A_investment)
# Here we initialize the ode function to solve
shock_model = Population_Solow_Model(fertility_rate=2.
 ↪1,solow_growth_parameters=solow_parameters_tech)
prior_model = Population_Solow_Model(fertility_rate=2.
 ↪1,solow_growth_parameters=solow_parameters)
shock_model.prep_model()
prior_model.prep_model()

count = 0
for k0 in IC:
    #Define initial conditions
    ts = np.linspace(0,100,501)
    pop_0 = np.exp(-.02*np.arange(life_expect))
    #solve the ODEs

    sol_tech = shock_model.solve(ts,pop_0,k0)
    sol_normal = prior_model.solve(ts,pop_0,k0)

    #Plot Capital
    color = colors[count]
    plt.plot(sol_tech.t,sol_tech.capital,'-',color = color)
    plt.plot(sol_normal.t,sol_normal.capital,'--',color = color)
    #Append legend info
    colorLine = Line2D([0,1],[0,1], linestyle='-', color=color)
    color_legend.append(colorLine)
    count = count + 1
#Make 2 legends, set other plot information
styleLine = Line2D([0,1],[0,1], linestyle='-', color="black")
style_legend.append(styleLine)
styleLine = Line2D([0,1],[0,1], linestyle='--', color="black")
style_legend.append(styleLine)
legend1 = plt.legend(color_legend, [r"$k_0$" + " = " + str(k0) for k0 in IC],␣
 ↪loc=3)
plt.legend(style_legend, label_list, loc=4)
plt.gca().add_artist(legend1)
plt.xlabel("Time (years)")
plt.ylabel("Capital k")
plt.title(f'Technology Investment Shock')
plt.ylim([0,10])
plt.show()
#fig2.savefig('Saved Figures/Tech Shock.pdf',bbox_inches = 'tight')
```



3.1 Importing Population Data from Augmented SIR

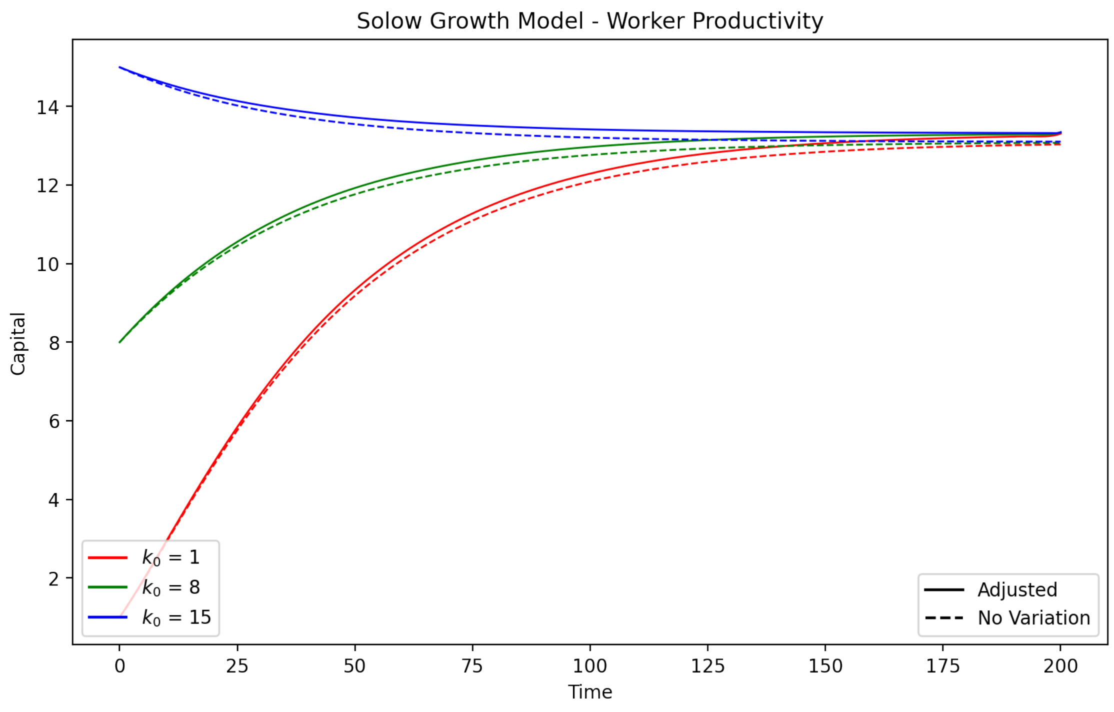
[ ]:
```
# Define constants
%store -r sol
N_matrix = sol.y
domain = sol.t
# Model for the U.S
alpha = 0.4                             # For developed countries
A = 1                                   # Assume constant technology
delta = 0.037                           # Depreciation rate of capital.
domain = sol.t
k0_list = [1, 8, 15]
t_span = (0,domain[-1])
s = 0.4

def n(t):
    if t not in domain: # If the input is not found in the domain, map it to␣
 ↪the nearest domain value
        t1 = np.argmin(la.norm(domain-t))
    else:
        t1 = t
    index = np.where(domain == t1)[0]
    return N_matrix[:,index[0]]
n = np.vectorize(n)


# Calculate the sum of the whole population
N = np.sum(N_matrix, axis=0)

lfgr = lambda t: 0.01 #I will assume that the Labor force growth rate is␣
 ↪constant, despite the population changes.
```

4 Worker Productivity

[ ]:
```
def Solow_productivity(t,k, p = np.array([1,1,1,1])):
    """Models the Solow growth curve, but with the variation that productivity␣
 ↪depends on population class
    Parameters:
    t (float) - The time element
    k (callable function) - Capital. This is the dependent variable
    """
    y = A * (k**alpha) * np.dot(p,n(t)) / (np.sum(n(t)))
    return s*y - (delta + lfgr(t))*k
```

[ ]:
```
p_list = [np.array([0.1, 0.8, 1.0, 0.3]), np.array([0.55, 0.55, 0.55, 0.55])]
label_list = ["Adjusted", "No Variation"]
```

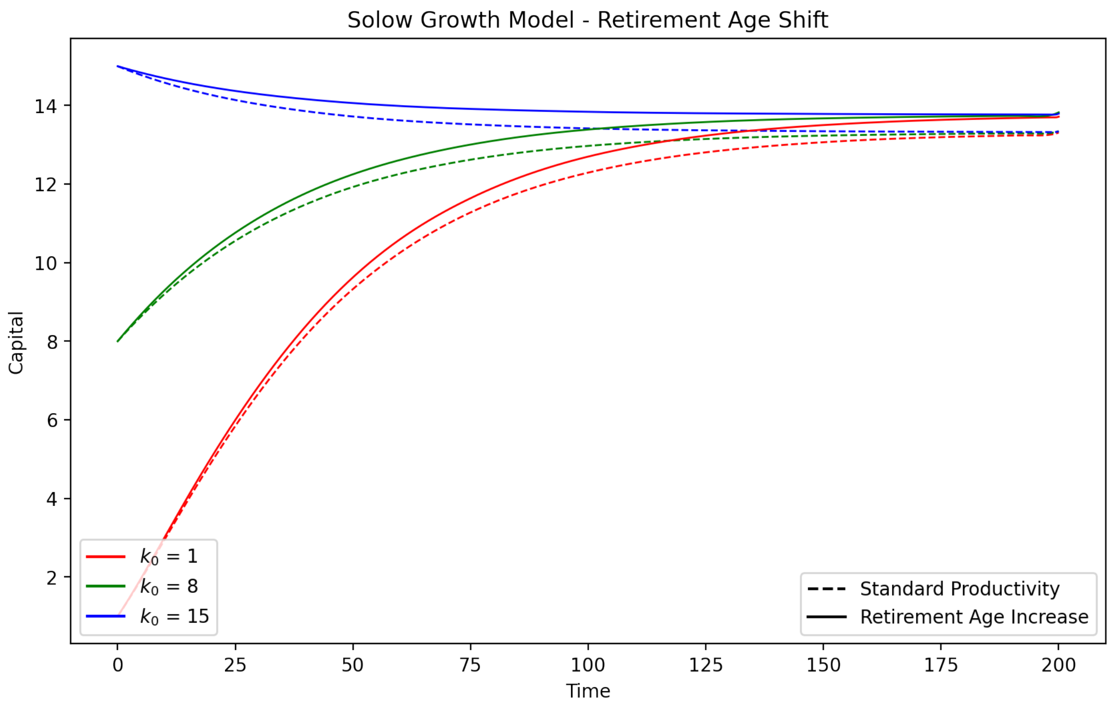
```
colors = ["red", "green", "blue"]
line_style = ["-", "--"]
color_legend = []
style_legend = []

fig = plt.figure(dpi=200,figsize=(10,6))

# To match up graphs with line styles and colors, I did a double for loop.
for p, lin in zip(p_list, line_style): # Iterates through the line styles
    for k0, color in zip(k0_list, colors): # Iterates through the colors
        # Plot the figure
        k = solve_ivp(Solow_productivity, t_span, [k0], t_eval=domain,␣
 ↪args=[p,])
        plt.plot(k.t, k.y[0], lin, color=color, linewidth=1)

        # Add data that simplifies the legend
        colorLine = Line2D([0,1],[0,1], linestyle='-', color=color)
        color_legend.append(colorLine)
    # The style line is used for the legend in the lower right of the graph
    # This line doesn't actually get plotted on the graph.
    styleLine = Line2D([0,1],[0,1], linestyle=lin, color="black")
    style_legend.append(styleLine)


legend1 = plt.legend(color_legend, [r"$k_0$" + " = " + str(k0) for k0 in␣
 ↪k0_list], loc=3)
plt.legend(style_legend, label_list, loc=4)
plt.gca().add_artist(legend1)
plt.title("Solow Growth Model - Worker Productivity")
plt.xlabel("Time")
plt.ylabel("Capital")
plt.show()

#fig.savefig("Worker Productivity.pdf")
```



5 Modeling a Retirement Age Increase

[ ]:
```
p_list = [np.array([0.1, 0.8, 1.0, 0.3]), np.array([0.1, 0.8, 1.0, 0.4])]
label_list = ["Standard Productivity", "Retirement Age Increase"]
colors = ["red", "green", "blue"]
line_style = ["--", "-"]
color_legend = []
style_legend = []

fig = plt.figure(dpi=200, figsize=(10,6))

for p, lin in zip(p_list, line_style):
    for k0, color in zip(k0_list, colors):
        k = solve_ivp(Solow_productivity, t_span, [k0], t_eval=domain,␣
 ↪args=[p,])
        plt.plot(k.t, k.y[0], lin, color=color, linewidth=1)

        # Add data that simplifies the legend
        colorLine = Line2D([0,1],[0,1], linestyle='-', color=color)
        color_legend.append(colorLine)

    styleLine = Line2D([0,1],[0,1], linestyle=lin, color="black")
    style_legend.append(styleLine)
```

```
legend1 = plt.legend(color_legend, [r"$k_0$" + " = " + str(k0) for k0 in␣
 ↪k0_list], loc=3)
plt.legend(style_legend, label_list, loc=4)
plt.gca().add_artist(legend1)
plt.title("Solow Growth Model - Retirement Age Shift")
plt.xlabel("Time")
plt.ylabel("Capital")
plt.show()
#fig.savefig("Retirement Age Shift.pdf")
```

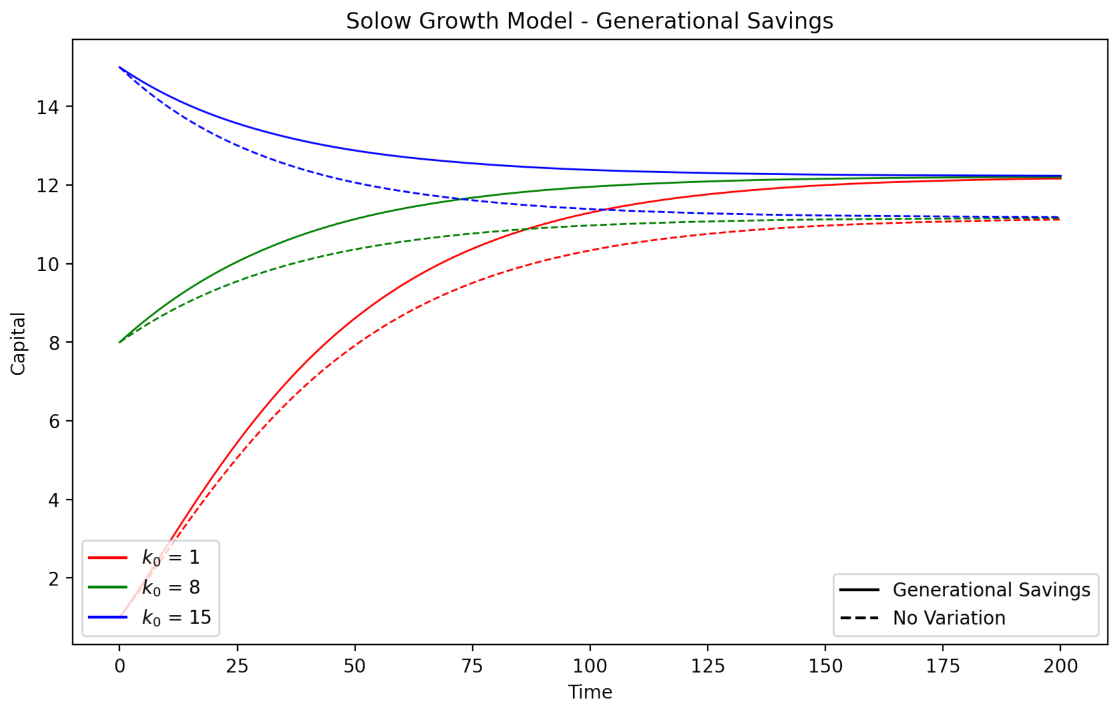


6 Savings Variation

```
[ ]: def Solow_savings(t,k, S = np.array([0.3,0.3,0.3,0.3])):
         """Models the Solow growth curve, but with the variation that productivity␣
      ↪depends on population class
         Parameters:
         t (float) - The time element
         k (callable function) - Capital. This is the dependent variable
         """
         y = A * (k**alpha)
         i = np.dot(S,n(t))*y / np.sum(n(t))
```

```
    return i - (delta + lfgr(t))*k
```

```
[ ]: s_list = [np.array([0.1, 0.3, 0.3, 0.1]), np.array([0.2, 0.2, 0.2, 0.2])]
     label_list = ["Generational Savings", "No Variation"]
     colors = ["red", "green", "blue"]
     line_style = ["-", "--"]
     color_legend = []
     style_legend = []

     fig = plt.figure(dpi=200,figsize=(10,6))

     for S, lin in zip(s_list, line_style):
         for k0, color in zip(k0_list, colors):
             k = solve_ivp(Solow_savings, t_span, [k0], t_eval=domain, args=[S,])
             plt.plot(k.t, k.y[0], lin, color=color, linewidth=1)

             # Add data that simplifies the legend
             colorLine = Line2D([0,1],[0,1], linestyle='-', color=color)
             color_legend.append(colorLine)

         styleLine = Line2D([0,1],[0,1], linestyle=lin, color="black")
         style_legend.append(styleLine)


     legend1 = plt.legend(color_legend, [r"$k_0$" + " = " + str(k0) for k0 in␣
      ↪k0_list], loc=3)
     plt.legend(style_legend, label_list, loc=4)
     plt.gca().add_artist(legend1)

     plt.title("Solow Growth Model - Generational Savings")
     plt.xlabel("Time")
     plt.ylabel("Capital")
     plt.show()
     #fig.savefig("Generational Savings.pdf")
```

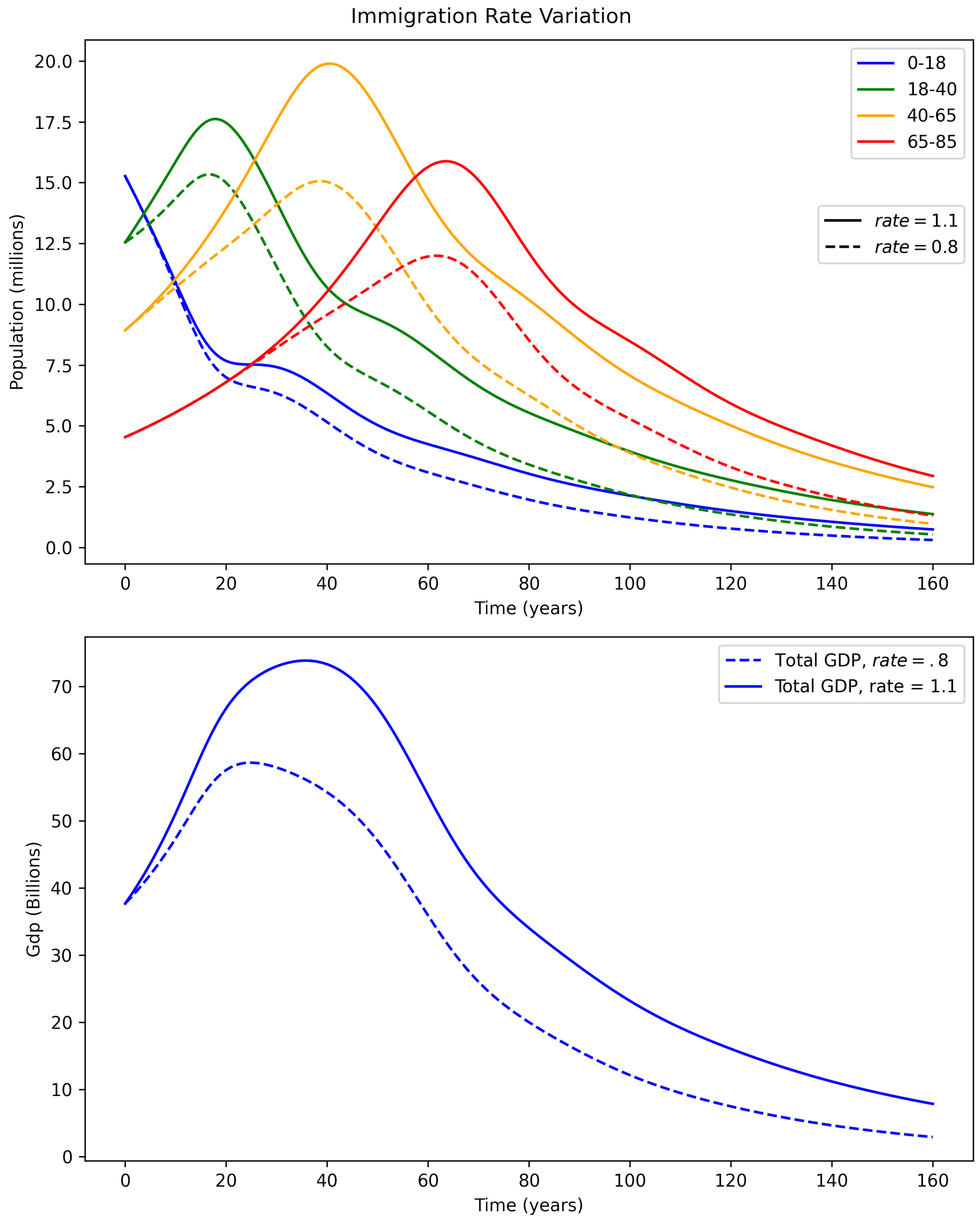


```
[ ]: life_expect = 85
     def fertility_rate(t):
         #Either a constant, or a function with respect to time
         return 2.1 + 1.9*(t>=20)-1.5*(t>=30) - .8*(t>=60)
     const_fert = 1.1
     solow_parameters = Solow_Model_Parameters()
     # Here we initialize the ode function to solve
     population_model =␣
      ↪Population_Solow_Model(fertility_rate=const_fert,life_expectancy=life_expect,solow_growth_p
      ↪imigration_rate=.8)
     population_model1 =␣
      ↪Population_Solow_Model(fertility_rate=const_fert,life_expectancy=life_expect,solow_growth_p
      ↪imigration_rate=1.1)

     ts = np.linspace(0,160,512)
     # Here we set the initial population
     x0 = np.exp(-.02*np.arange(life_expect))

     population_model.prep_model()
     population_model1.prep_model()
     sol = population_model.solve(ts,x0,3)
     sol1 = population_model1.solve(ts,x0,3)
     # Here consolidate the ages into age brackets, to make the data more visible
     # Grouping the age brackets will also make it easier for our model
```

```
values,labels = consolidate_age_groups(sol.population,return_labels=True)
values1,labels1 = consolidate_age_groups(sol1.population,return_labels=True)
colors = ['blue', 'green', 'orange', 'red']

# Here we plot the results
print("With capital this time as well.")
plt.figure(figsize=(8,10), dpi=300)
plt.subplot(211)
# plt.plot(sol.t,np.ones_like(sol.t)*const_fert,":k", color='red', label =␣
↪"fertility rate")
for i in range(len(colors)):
    plt.plot(sol.t,values[i], color=colors[i],linestyle='--')
    plt.plot(sol1.t,values1[i], color=colors[i], label = labels[i])
#define 2 legends
legend1 =plt.legend(loc='upper right', bbox_to_anchor=(1., 1))
#Make 2 legends, set other plot information
style_legend = []
label_list = ["$rate = 1.1$", "$rate = 0.8$"]
styleLine = Line2D([0,1],[0,1], linestyle='-', color="black")
style_legend.append(styleLine)
styleLine = Line2D([0,1],[0,1], linestyle='--', color="black")
style_legend.append(styleLine)
plt.legend(style_legend, label_list, loc='upper right', bbox_to_anchor=(1., 0.
↪7))
plt.gca().add_artist(legend1)


plt.xlabel("Time (years)")
plt.ylabel("Population (millions)")

plt.subplot(212)

# plt.ylim([0, 20])
plt.legend()
plt.plot(sol.t,sol.y*np.sum(sol.population,axis = 0), '--', color = 'blue',␣
↪label = "Total GDP, $rate = .8$")
plt.plot(sol1.t,sol1.y*np.sum(sol1.population,axis = 0), color = 'blue', label␣
↪= "Total GDP, rate = 1.1")
plt.xlabel("Time (years)")
plt.ylabel("Gdp (Billions)")

plt.legend()

plt.suptitle('Immigration Rate Variation')
plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

```
With capital this time as well.
```


Immigration Rate Variation
$rate = 1.1$
$rate = 0.8$
Population (millions)
Time (years)
Total GDP, $rate = .8$
Total GDP, rate = 1.1
Gdp (Billions)
Time (years)