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
```

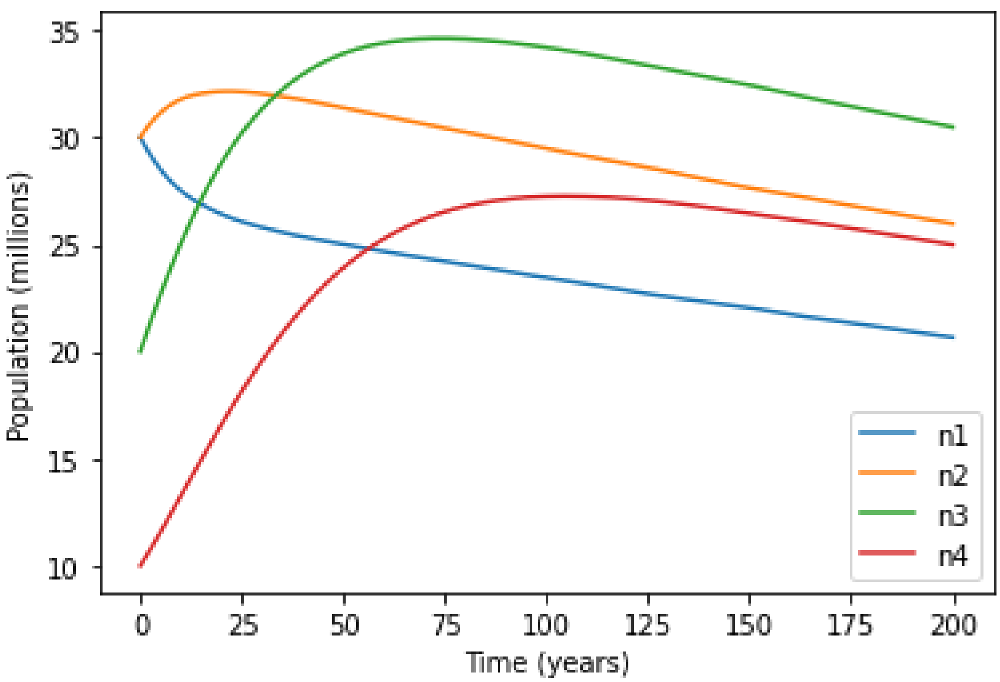
1 Population Augmented-SIR Model

[ ]:
```
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)
```

[ ]:
```python
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):
```

```python
            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:
```

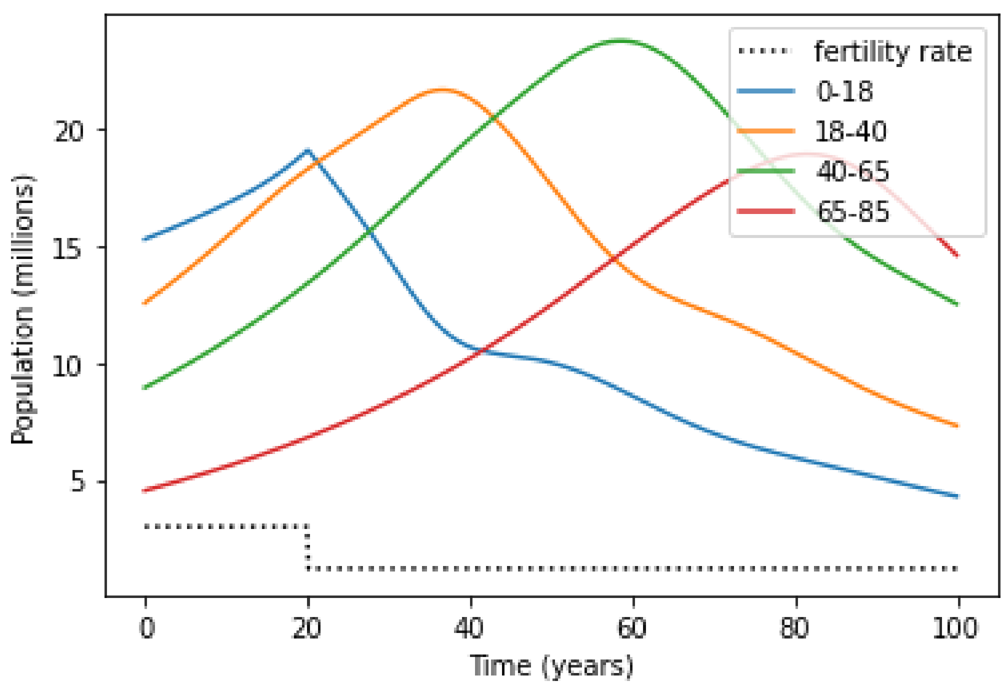
```
            - dx: Array representing the rates of change of the state variables.
            """
            dx = np.zeros_like(x)

            # 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
```

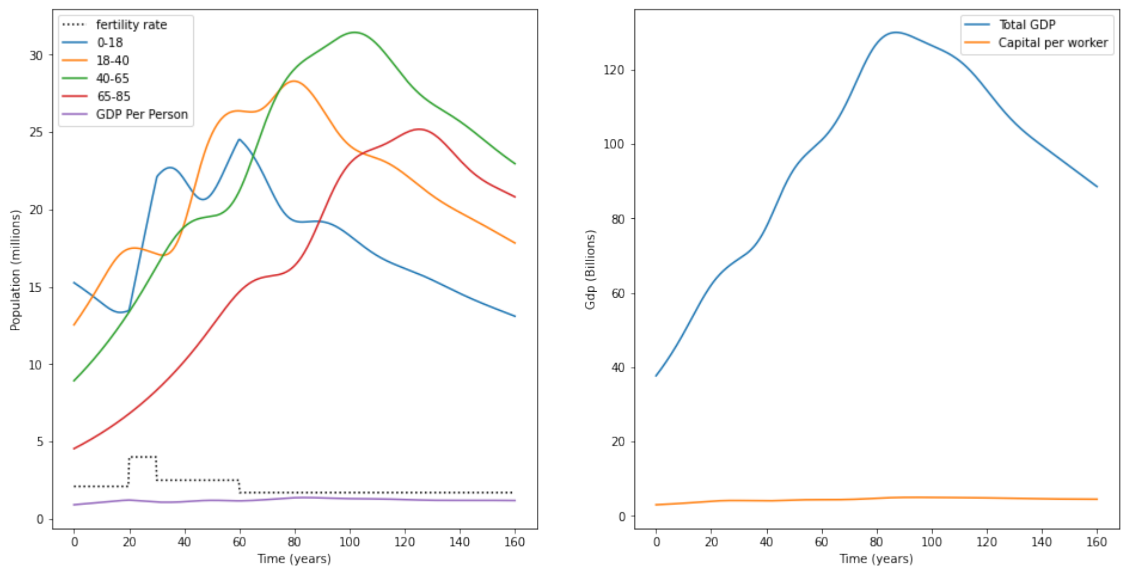
[ ]:
```
life_expect = 85
def fertility_rate(t):
    #Either a constant, or a function with respect to time
    return 1.2 + 1.8*(t<=20)
# Here we initialize the ode function to solve
```

```
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.
```

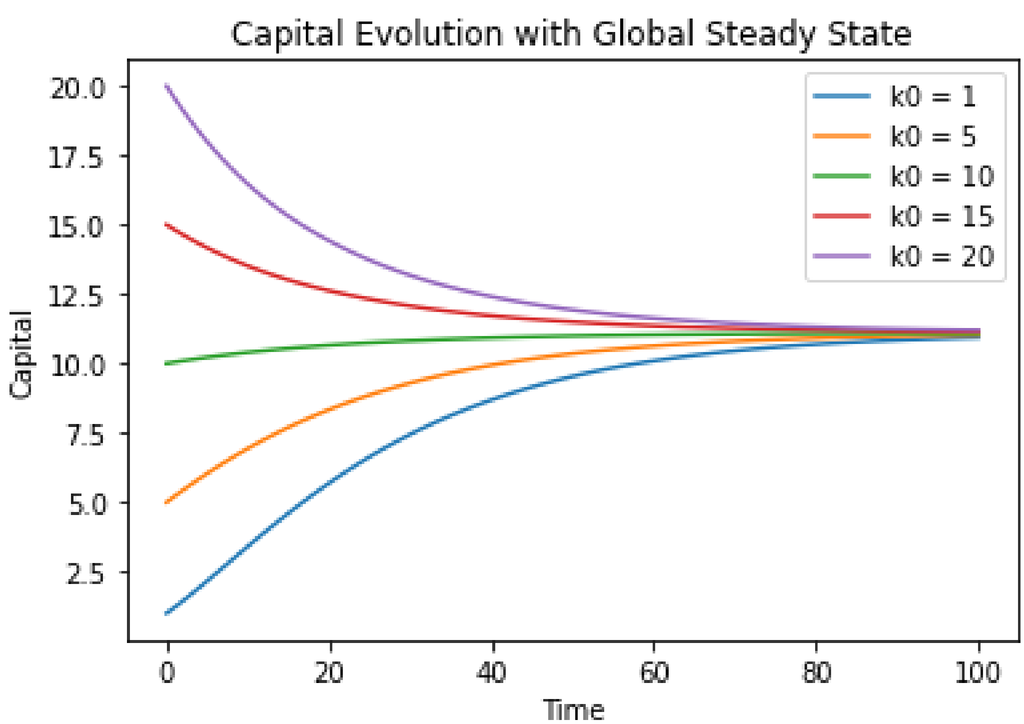


2 Baseline Solow Growth Model

[ ]:
```
#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
```

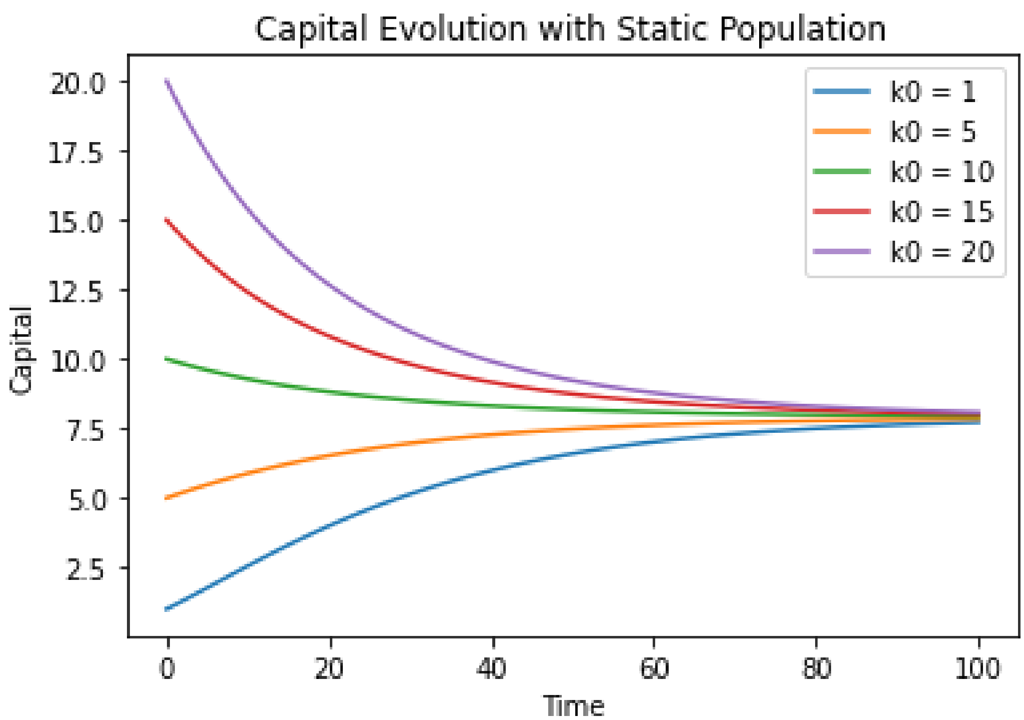
[ ]:
```
#A very simple model where capital trends towards a global steady state
def ksolve(t, k, alpha=0.5, s=0.3, delta=0.08, n=0.01, A=1):
    return kprime(k, alpha=0.5, s=0.3, delta=0.08, n=0.01, A=1)

#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):
```

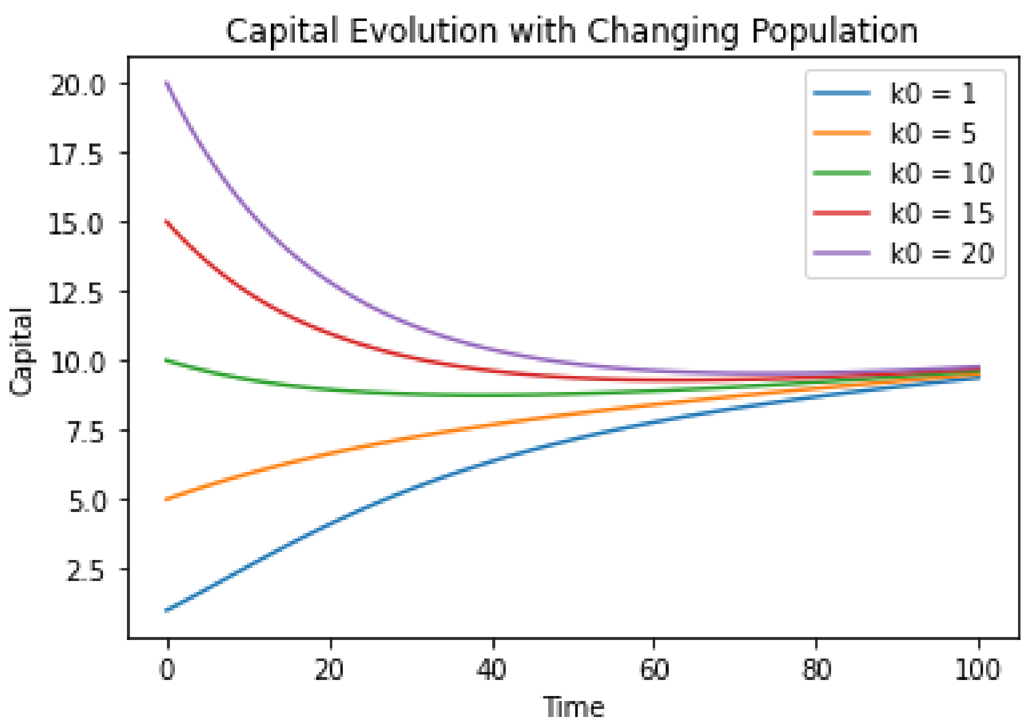
```
    #N is a vector with the breakdown of population by age
    #C is a vector with the contribution of each age demographic

    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)
```

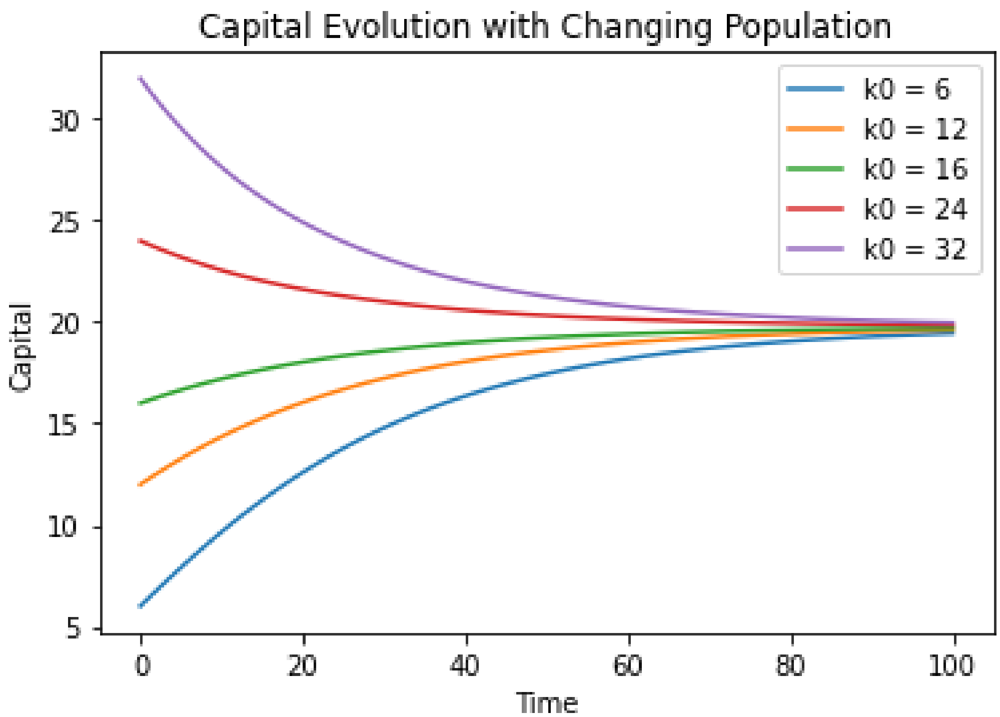
```python
sol1 = solve_ivp(ksolve_pop2, t_span, k1, t_eval=T)
sol2 = solve_ivp(ksolve_pop2, t_span, k2, t_eval=T)
sol3 = solve_ivp(ksolve_pop2, t_span, k3, t_eval=T)
sol4 = solve_ivp(ksolve_pop2, t_span, k4, t_eval=T)


#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]
```

[ ]:
```python
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

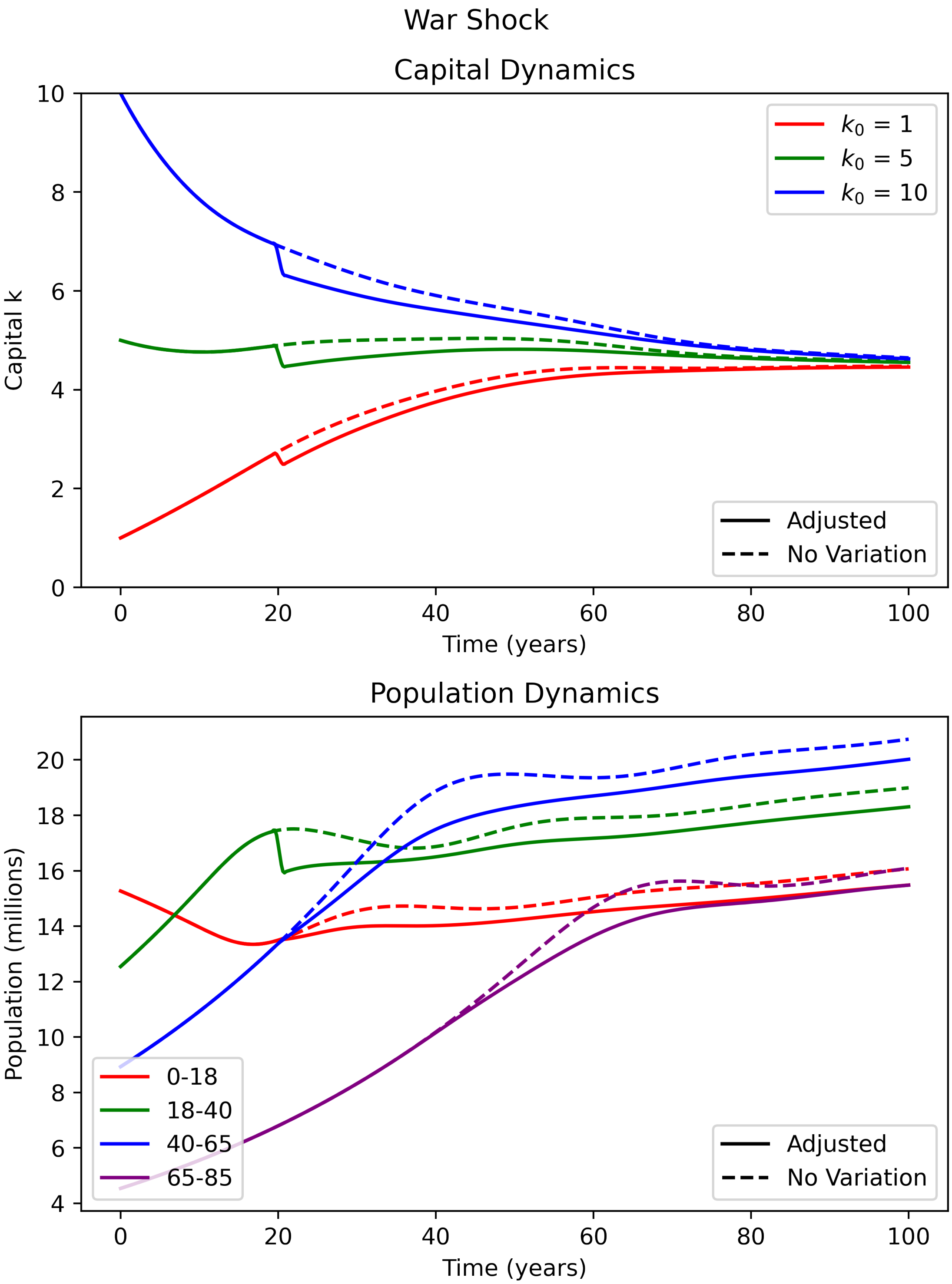
[ ]:
```
#Define several k initial conditions, plotting conditions
IC = [1,5,10]
count = 0
fig1 = plt.figure(figsize=(6,8),dpi=500)
plt.subplot(211)
label_list = ["Adjusted", "No Variation"]
colors = ["red", "green", "blue"]#,"orange","purple"]
line_style = ["--", "-"]
color_legend = []
style_legend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

```python
#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')
```

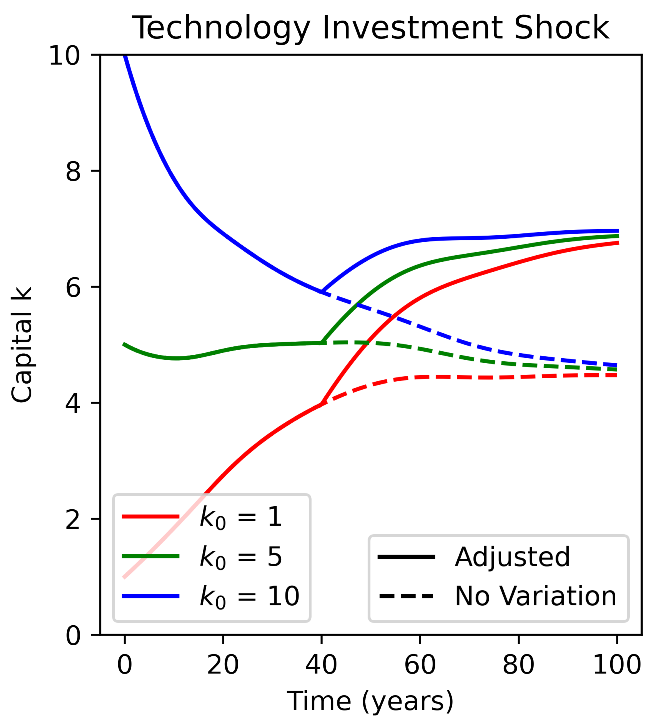
```
/tmp/ipykernel_14754/1748665256.py:44: RuntimeWarning: invalid value encountered
in double_scalars
  return A * k ** self.alpha * n
```


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
```

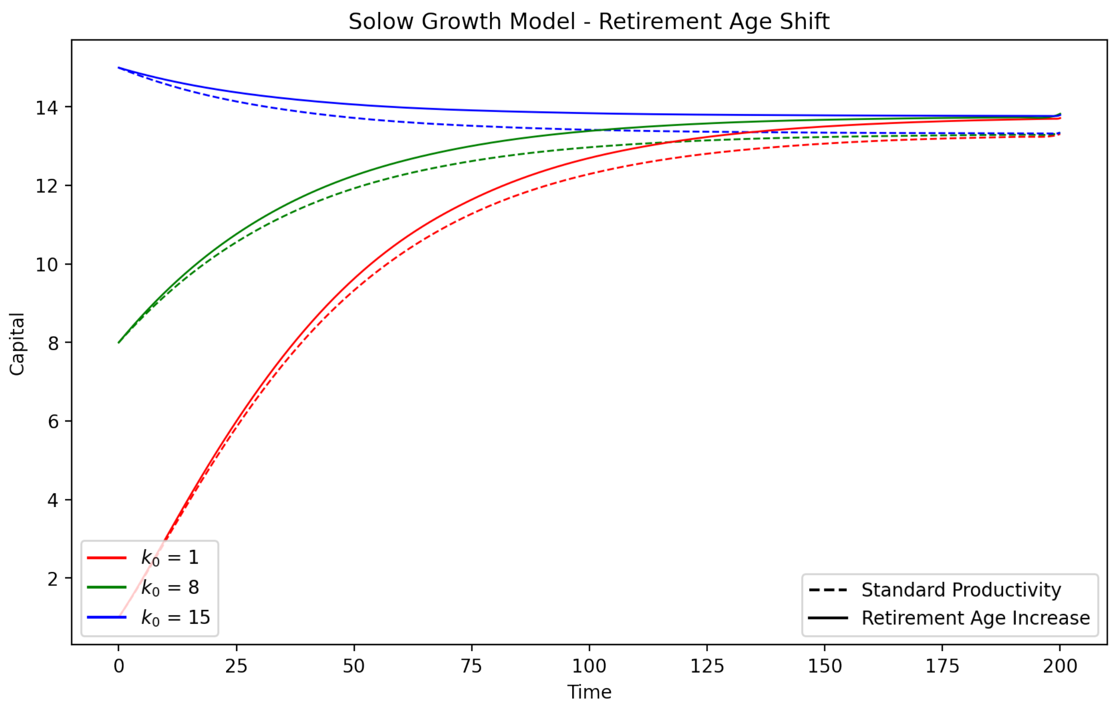
[ ]:
```
p_list = [np.array([0.1, 0.8, 1.0, 0.3]), np.array([0.55, 0.55, 0.55, 0.55])]
label_list = ["Adjusted", "No Variation"]
```

```
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")
```

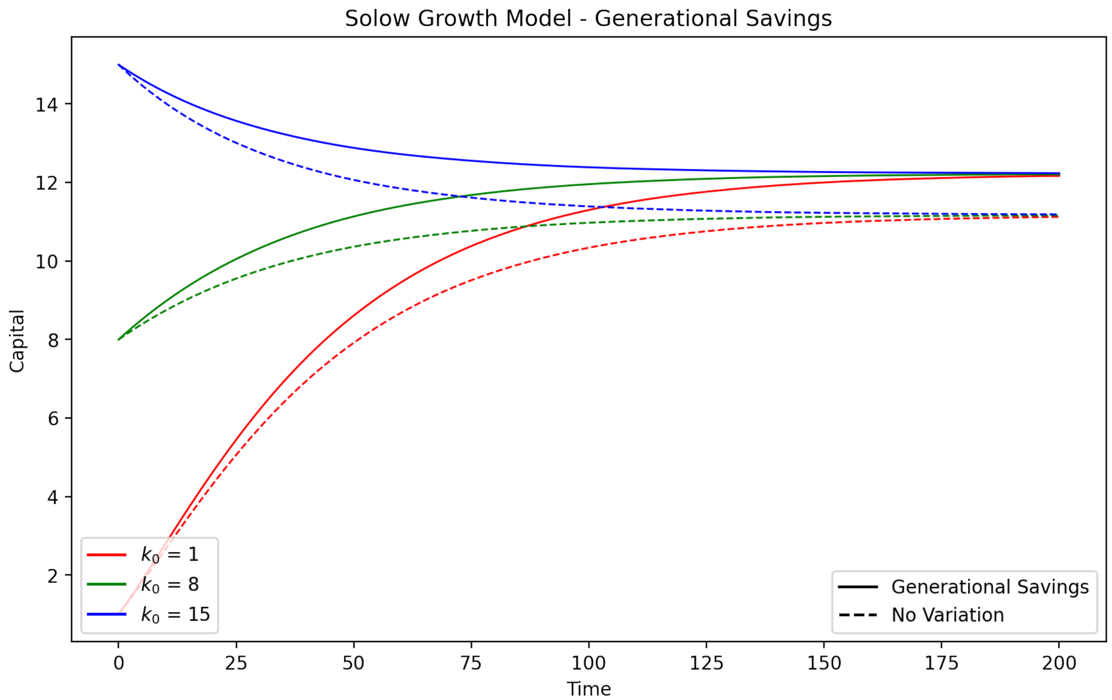


6 Savings Variation

```
[ ]: 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.
```