Python Implementation of MPW (2023)

May 4, 2023

1 MPW (2023) MODEL SIMULATION

Primary Author: John Mantus

Last edited: 5/4/23 by JM

This file generates the projection estimates from *A Unified Long-Run Macroeconomic Projection of Health Care Spending, the Federal Budget, and Benefit Programs in the US* by John Mantus, Gaobo Pang, and Mark J. Warshawsky, published as an AEI Working Paper on 05/01/2023.

This is also the underlying code for the interactive dashboard found here: *insert link if/when page goes live*

Abstract
In the official reduced form models used by the CBO, Treasury, and the Social Security and Medicare Trustees for projections and policy analysis, many key variables, like real interest rates, health care cost and economic growth, are assumed, often based on a continuation of past trends. By contrast, in our model, these variables are outcomes produced by supply and demand, based on logical functional forms and deep parameter estimates from the literature or empirical analysis. This latter approach provides a more credible and realistic basis for projections with changing underlying conditions and less subject to optimistic hopefulness. We find that, within the next five to ten years, the deficit relative to national income will grow significantly beyond historical experience, and with the prospect of continued increases, without policy changes, may be regarded as unsustainable. In particular, we project that debt-to-GDP will be 132 percent in 2032, compared to CBO's recent projection of 115 percent. In 2052, we project 258 percent compared to CBO's 189 percent. Our projection of national health care spending relative to GDP in 2072 is 29.9 percent, compared to 28.4 percent by CMS, used by the Medicare Trustees. Longer term, our projections show even worse outcomes for debt, deficits, Social Security, Medicare, and health care spending than official projections, and declines in the rate of welfare improvement. These results mainly owe to higher rates of health care inflation, arising from labor shortage effects because health care is produced in a low productivity, labor-dependent sector, and real interest rates rise in the long-run because of increasing debt, bringing about the vicious cycle of growing deficits and debt.

[12]:
```
# Import packages needed for computations
import numpy as np # Handy for math
import sympy as sp # Can add precision and clarity to equations (not used␣
↪currently)
import pandas as pd # For data storage
from scipy.optimize import fsolve # For solving systems of equations
```

2 Computing Health Care Elasticities

We include two health care elasticities; one with respect to income, the other with respect to the relative price of health care, as compared to all other goods.

Baseline
- Income elasticity begins at 1.2, declines to 1.1 over first ten years, declines further to 1.0 over next fifteen years, then declines to 0.9 over the next 25 years (50 years total), then is constant.
- Price elasticity declines from -0.5 to -0.56 over first ten years, then remains constant.

Alternative
- Income elasticity begins at 1.2, then changes linearly to the new value over the initial 10 years, then remains constant.
- Price elasticity changes from -0.5 to the new value over first ten years, then remains constant.

[10]:
```
# Below are the functions used to compute changes in the elasticities, described␣
↪above
# Baseline computations
def compute_inc_el_base(inc_el, fin_inc_el):
    for t in range(len(years)):
        if t < period1:
            df_el.loc[t+1, 'Income_Elasticity'] = df_el.loc[t,␣
↪'Income_Elasticity']-(inc_el-fin_inc_el)/(3*period1)
        elif t < period1 + period2:
            df_el.loc[t+1, 'Income_Elasticity'] = df_el.loc[t,␣
↪'Income_Elasticity']-(inc_el-fin_inc_el)/(3*period2)
        elif t < period1 + period2 + period3:
            df_el.loc[t+1, 'Income_Elasticity'] = df_el.loc[t,␣
↪'Income_Elasticity']-(inc_el-fin_inc_el)/(3*period3)
        else:
            df_el.loc[t+1, 'Income_Elasticity'] = df_el.loc[t,␣
↪'Income_Elasticity']

def compute_pri_el(pri_el, fin_pri_el):
    for t in range(len(years)):
        if t < period1:
            df_el.loc[t+1, 'Price_Elasticity'] = df_el.loc[t,␣
↪'Price_Elasticity']-(pri_el-fin_pri_el)/(period1)
        else:
            df_el.loc[t+1, 'Price_Elasticity'] = df_el.loc[t, 'Price_Elasticity']

# Alternative
def compute_inc_el_new(inc_el, fin_inc_el):
    for t in range(len(years)):
        if t < 10:
```

```
            df_el.loc[t+1, 'Income_Elasticity'] = df_el.loc[t,␣
 ↪'Income_Elasticity']-(inc_el-fin_inc_el)/(period1)
        else:
            df_el.loc[t+1, 'Income_Elasticity'] = df_el.loc[t,␣
 ↪'Income_Elasticity']
```

[7]:
```
# Computing and storing the health care elasticities in a DataFrame
df_el = pd.DataFrame(columns=['Year','Elasticity_Adj_Factor',␣
 ↪'Income_Elasticity', 'Price_Elasticity'])
df_el['Year'] = years
df_el.loc[0, 'Income_Elasticity'] = inc_el
df_el.loc[0, 'Price_Elasticity'] = pri_el

period1 = 10 # Years
period2 = 15
period3 = 25

# Currently detecting 'Alternative' based on income elasticity choice since␣
 ↪price elasticity slider has not been implemented to dashboard
if fin_inc_el == 0.9: # Baseline
    compute_inc_el_base(inc_el = inc_el, fin_inc_el = fin_inc_el)
    compute_pri_el(pri_el = pri_el, fin_pri_el = fin_pri_el)
else:                 # Alternative
    compute_pri_el(pri_el = pri_el, fin_pri_el = fin_pri_el)
    compute_inc_el_new(inc_el = inc_el, fin_inc_el = fin_inc_el)
```

3 Demographics

Demographic projections from the Congressional Budget Office (CBO) play a large role in our projections. Health care spending differs greatly by age and sex, with, e.g., women over the age of 65 spending more than any other group. Further, labor force participation varies with age and sex, as well.

[25]:
```
# Import CBO Demographic Projections --- Both for HC spending matrix and LFPR
df_cbo_hc = pd.read_excel(r'/Data/Raw/proj_by_year_group_feb23.xlsx')
df_cbo_lf = pd.read_excel(r'/Data/Raw/proj_by_year_group_lfpr_feb23.xlsx')

df_cbo_hc['2021'] = df_cbo_hc['2022']
df_cbo_lf['2021'] = df_cbo_lf['2022']

df_cbo_hc = df_cbo_hc.rename(columns={'age_group': 'agegroup'})

# Import computed spending matrix
pred_hc_exp = pd.read_excel(r'/Data/Raw/2020_phc_per_capita.xls')
```

```
pred_hc_exp = pred_hc_exp.rename(columns={'gender': 'sex'})

# Clean for later merge
pred_hc_exp['sex'] = pred_hc_exp['sex'].str.replace('1 MALE', 'Male').str.
 ↪replace('2 FEMALE', 'Female')

# Pull spending matrix computed with 2020 MEPS
pred_hc_exp_2020 = pred_hc_exp[pred_hc_exp.year == 2020]

# Pull variables of interest: Spending amount, sex, age group, and payer
df_hc_exp = pred_hc_exp_2020.loc[:, ['agegroup', 'sex', 'payer',␣
 ↪'cms_pred_per_capita']]
df_hc_exp = pd.merge(df_hc_exp, df_cbo_hc, on=['agegroup', 'sex'])

# Import assumped LFPR and unemployment matrix from BLS Dec 2021
lfpr_dec21 = pd.read_excel(r'/Data/Raw/lfpr_dec_21.xlsx')
```

[2]:
```
# Collect projections of 65+ population for r+ calculation
plus_65 = df_cbo_hc.loc[(df_cbo_hc['agegroup'] == '65-84') |␣
 ↪(df_cbo_hc['agegroup']== '85+')].sum(numeric_only=True, axis=0)
plus_65.name = '65_plus'
total = df_cbo_hc.sum(numeric_only=True, axis=0)
total.name = 'Total'

pop=pd.concat([plus_65.sort_index(), total.sort_index()], axis=1)
pop['Year'] = pop.index
pop['Eld Share'] = pop['65_plus']/pop['Total']
pop['Delta'] = pop['Eld Share'].diff() # This is the value used in computing␣
 ↪r_plus
pop['Delta'].iloc[0] = 0 # Replaces missing 2021 value with 0
```

Our labor force projections use the demographic data mentioned above, as well as the Dec 2021 labor force participation and unemployment data from the Bureau of Labor Statistics. Further, we include an assumption of declining hours worked, consistent with the most recent Social Security Trustees Report.

[4]:
```
# Compute labor force projections
labor_df = pd.DataFrame(columns = ['Year','Labor_Force', 'Hours_per_Week',␣
 ↪'Labor_Hours'])
labor_df['Year'] = years

# For each year, compute both the total number of individuals in the labor␣
 ↪market and the number of hours they will work in a week, on average.
for year in years:
    temp_sum = 0
    for s in lfpr_dec21.Sex.unique(): # For each sex
        for age_grp in lfpr_dec21['Age Group'].unique(): # For each age group
```

```
            temp = pd.DataFrame()
            temp = lfpr_dec21[(lfpr_dec21['Sex'] == s) & (lfpr_dec21['Age␣
↪Group'] == age_grp)]
            lfpr = temp['LFPR, 2021 BLS'][min(temp.index)] # Pull lfpr
            employment = temp['Employment'][min(temp.index)] # Pull employment
            pop_df = df_cbo_lf[(df_cbo_lf['sex'] == s) & (df_cbo_lf['age_group']␣
↪== age_grp)][str(year)] # Pull population for age group and sex
            pop_ = pop_df[min(pop_df.index)]
            temp_sum += lfpr*employment*pop_/1000000 # Compute labor force, in␣
↪millions of workers

    labor_df.loc[labor_df['Year'] == year, 'Labor_Force'] = temp_sum # Store the␣
↪year's total value

    # Compute hours worked
    if year == min(years):
        labor_df.loc[labor_df['Year'] == year, 'Hours_per_Week'] = hrs_0 #␣
↪Initial hours worked per week
    else:
        labor_df.loc[labor_df['Year'] == year, 'Hours_per_Week'] =␣
↪hrs_0*(1-hrs_dec)**(year-min(years)) # Annual decline in hours worked.

# Compute total number of labor hours for a given year, in trillions of hours
labor_df['Labor_Hours'] = labor_df['Labor_Force']*labor_df['Hours_per_Week']*52/
↪1000000
```

4 Growth Factors

We include several growth factors in our projections.
g_1 is the growth in labor-augmenting technical progress in the Cobb-Douglas formulation for all other output.
g_2 is the annual deepening of health care capital, assumed to be zero in our base analysis but becomes slightly positive in alternate specifications.
g_3 is the growth in labor productivity in the Leontief health care production formulation.

[8]:
```
# Compute growth factors
g_df = pd.DataFrame(columns = ['Year', 'g_1', 'g_2', 'g_3'])
g_df['Year'] = years

for year in years:
    g_df.loc[g_df['Year'] == year, 'g_1'] = (1+g1)**(year-min(years))
    g_df.loc[g_df['Year'] == year, 'g_2'] = (1+g2)**(year-min(years))
    g_df.loc[g_df['Year'] == year, 'g_3'] = (1+g3)**(year-min(years))
```

4.1 Simulate

Now we are able to simulate our model. In practice, this means solving several systems of equations each year. The order indicates that that later calculations depend upon earlier calculations (e.g., tax revenue depends on wages).
For 2021, we solve 4 separate systems in the following order:

i. Production functions
i. First-Order Conditions (FOCs)
ii. Income Identities
iii. Federal Government Sector

In future years, there are interactions between these systems, thus they cannot be solved so easily. In addition, we must incorporate projected changes to health care demand and labor force through demographics, as well as capital stocks based on last year's stock and investment and depreciation. The order, thus, is as follows:

i. Labor Force, Capital, and Health Care Spending
ii. First-Order Conditions (FOCs)
iii. Production Functions
iv. Income Identities and Federal Government Sector

Below you can find the complete specification of the model (also found in *Appendix A* of the original paper):

Production Functions
- $f_{1t} = \alpha_1(g_1^t L_{1t})^{1-\beta_1}(K_{1t}^{\beta_1})$
- $f_{2t} = min(\frac{\hat{L_{2t}} g_3^t}{\beta_2}, \frac{\hat{K_{2t}}}{\beta_3})$

Income Identitities
- $Y_t = f_{1t} + p_t f_{2t}$
- $\hat{f_{2t}} = F(demographics, income, prices) = min(\frac{\hat{L_{2t}} g_3^t}{\beta_2}, \frac{\hat{K_{2t}}}{\beta_3})$
- $Y_t = C_t + I_t + G_t$
- $C_t = Y_t - I_t - G_t$
- $I_t = a(r_t^+)Y_t$, with an interest elasticity of -0.2
- $G_t = Def_t(Y_t) + ND_t(population)$
- $Y_t = w_t L_t + r_t K_t$

Factors of Production
- $K_t = (1-b)K_{t-1} + I_{t-1}$
- $L_t = F(demographics, LFPR, hours, unemployment)$

First Order Conditions (FOCs)
- $w_t = (1-\beta_1)\alpha_1 g_1^t [\frac{K_t - \hat{K_{2t}}}{g_1^t(L_t - \hat{L_{2t}})}]^{\beta_1}$
- $r_t = \beta_1 \alpha_1 g_1^t [\frac{K_t - \hat{K_{2t}} g_2^t}{g_1^t(L_t - \hat{L_{2t}})}]^{\beta_1 - 1}$
- $p_t = \frac{A_t}{\hat{f_{2t}}}$, where

- $A_t = \alpha_1(g_1^t(1-\beta_1)(B_t)^{\beta_1}L_t + \beta_1(B_t)^{\beta_1-1}K_t - B_t^{\beta_1}(g_1^t(L_t - \hat{L_{2t}})))$, and
- $B_t = \frac{K_t - \hat{K_{2t}}}{g_1^t(L_t - \hat{L_{2t}})}$

Federal Government Sector
- $Df_t = G_t - T_t + Bf_t + R_t$
- $T_t = \tau Y_t + \pi(w_t L_t - EGHI_t) + 0.05SS_t$
- $Bf_t = SS_t + GHC_t - 0.39p_t * Medicaid - 0.56p_t * All\_Other$
- $SS_t = l * i * rep * ret\_pop * total\_labor\_income$
- $GHC_t = p_t * (Medicare + Medicaid + All\_Other)$
- $R_t = D_t(r_t^+ - rp_t)$
- $D_t = D_{t-1} + Df_{t-1}$, $D_0 = 22.3$
- $r_t^+ = r_t + 0.045(\frac{D_t}{Y_t} - \frac{D_{t-1}}{Y_{t-1}}) - 0.57 * 0.35(\frac{D_t}{Y_t} - \frac{D_{t-1}}{Y_{t-1}}) - 0.165 * 0.10(\frac{D_t}{Y_t} - \frac{D_{t-1}}{Y_{t-1}}) - 1.293 * (elderly_t - elderly_{t-1})$

[10]:
```
# Systems for Initial Year

# Initial Production Functions
def production_init(x):
    f_1 = x[0]
    f_2 = x[1]
    Y = x[2]
    K = x[3]

    F = np.empty((4))

    # Production Functions
    F[0] = alpha1*(((g_1*L_1)**(1.0-beta1))*(K_1)**beta1) - f_1
    F[1] = (g_2*K_2)/(beta3) - f_2

    # Income Identity
    F[2] = f_1 + p_0*f_2 - Y

    F[3] = K_0 - K

    return F

# Initial first order conditions (FOCs)
def foc_init(x):
    L = labor_df.loc[labor_df['Year'] == year, 'Labor_Hours'] # Labor hours␣
 ↪computed from CBO and BLS matrices

    w = x[0]
    r = x[1]
    p = x[2]
    A = x[3]
    B = x[4]
```

```
    F = np.empty((5))

    # First Order Conditions
    F[0] = (1-beta1)*alpha1*g_1*((K-K_2)/(g_1*(L-L_2)))**beta1 - w
    F[1] = beta1*alpha1*((K-K_2*g_2)/(g_1*(L-L_2)))**(beta1-1) - r
    F[2] = A/f2_hat - p
    F[3] = alpha1*(g_1*(1.0-beta1)*(B)**beta1*L + beta1*(B)**(beta1-1.0)*K -␣
 ↪B**beta1*(g_1*(L-L_2))) - A
    F[4] = (K-K_2)/(g_1*(L-L_2)) - B

    return F

# Initial income identities
def identities_init(x):
    C = x[0]
    I = x[1]
    G = x[2]

    F = np.empty((3))

    # Income Identitities
    F[0] = Y_use - I - G - C
    F[1] = a*Y_use - I
    F[2] = Def + NDef - G

    return F

# Initial federal government sector
def fed_gov_init(x):
    L = labor_df.loc[labor_df['Year'] == year, 'Labor_Hours'] # Labor hours␣
 ↪computed from CBO and BLS matrices
    Df, T, SS, Bf, R, rp, r_gov, D, r_plus, a = x

    F = np.empty((10))

    F[0] = G_use-T+Bf+R - Df
    F[1] = tau*Y_use+pi*(w_use*L-EGHI_use)+tau_ss*SS_use - T
    F[2] = SS_use - SS
    F[3] = Bf_use - Bf
    F[4] = D*(r_plus-rp) - R
    F[5] = rp_c*r_plus - rp
    F[6] = r_plus - rp - r_gov
    F[7] = D_0 - D
    F[8] = r_use + debt_int*(D/Y_use - D_lag/Y_lag) - for_int*(for_share*(D/
 ↪Y_use - D_lag/Y_lag))-fed_int*(fed_share*(D/Y_use - D_lag/Y_lag)) -␣
 ↪eld_int*(eld_del-eld_del_lag) - r_plus
```

```
    F[9] = a_0 - a

    return F

##########################################################################
# Systems for every other year

# First order Conditions
def foc(x):
    w, r, p, A, B = x

    F = np.empty((5))

    # First Order Conditions
    F[0] = (1-beta1)*alpha1*g_1*((K_-K_2/g_2)/(g_1*(L-L_2)))**beta1 - w
    F[1] = beta1*alpha1*((K_-K_2*g_2)/(g_1*(L-L_2)))**(beta1-1) - r
    F[2] = A/f2_hat - p
    F[3] = alpha1*(g_1*(1-beta1)*((B)**beta1)*L + beta1*(B)**(beta1-1)*K_ -␣
 ↪B**beta1*(g_1*(L-L_2))) - A
    F[4] = (K_-K_2)/(g_1*(L-L_2)) - B

    return F

# Production of Healthcare and All Other output
def productions(x):
    f_1 = x[0]
    f_2 = x[1]
    Y = x[2]
    K = x[3] # Only included for convenience - computed elsewhere as K_

    F = np.empty((4))

    # Production Functions
    F[0] = alpha1*(((g_1*L_1)**(1.0-beta1))*(K_1)**beta1) - f_1
    F[1] = (g_2*K_2)/(beta3) - f_2

    # Income Identity
    F[2] = f_1 + p*f_2 - Y

    F[3] = K_ - K

    return F

# Federal government sector and Income Identities solved simultaneously, relying␣
 ↪on previous computations
def fed_gov_ids(x):
    C, I, G, Df, T, SS, Bf, R, rp, r_gov, D, r_plus, a = x
```

```
    F = np.empty((13))

    # Income Identitities
    F[0] = Y - I - G - C
    F[1] = a*Y - I
    F[2] = Def + NDef - G

    # Federal Government
    F[3] = G-T+Bf+R - Df
    F[4] = tau*Y+pi*(w*L-EGHI)+tau_ss*SS - T
    F[5] = ss_i*ss_l*rep*w*52*hrs_wk*plus_65/1000000000000 - SS
    F[6] = fed_ghc*p + SS - Bf
    F[7] = D*(r_gov) - R
    F[8] = rp_c*r_plus - rp
    F[9] = r_plus - rp - r_gov
    F[10] = D_lag + Df_lag - D
    F[11] = r + debt_int*(D/Y - D_0/Y_init) - for_int*(for_share*(D/Y - D_0/
 ↪Y_init))-fed_int*(fed_share*(D/Y - D_0/Y_init)) -␣
 ↪eld_int*(eld_del-eld_del_lag) - r_plus
    F[12] = a_0*(1+int_el*(r_plus-r_plus_init)/(r_plus_init)) - a

    return F
```

[5]:
```
# The code below actually computes the variables of interest. It stores final␣
 ↪projections in a DataFrame called 'projections'

# The code below initializes several variables that will be updated annually␣
 ↪later.
year = min(years)
# Where projections will be stored
projections = pd.DataFrame(columns=['Year', 'f_1', 'f_2', 'Y', 'K',
                                    'w', 'r', 'p', 'A', 'B', 'C', 'I', 'G',␣
 ↪'Df', 'T', 'SS', 'Bf', 'R',
                                    'rp', 'r_gov', 'D', 'r_plus', 'a'])
# Growth factors
g_1 = g_df.loc[g_df['Year'] == year, 'g_1']
g_2 = g_df.loc[g_df['Year'] == year, 'g_2']
g_3 = g_df.loc[g_df['Year'] == year, 'g_3']

# Factors of Production
L, L_1, L_2 = L_0, L1_0, L2_0
K, K_1, K_2 = K_0, K1_0, K2_0

# Initial values for 2021.
Def = 0.9324
NDef = 0.7051
```

```
f2_hat = 4.255195243655630000
p_0 = 1.0

a = a_0

NDef = non_def_0

K_lag = K
I_lag = 0

f2_hat_lag = 4.255195243655630000
Medicaid = medicaid_0
Medicare = medicare_0
Other_hc = other_0
eld_pop = eld_pop_0
hrs_wrk = hrs_0

D_lag = D_0
Df_lag = 0.0

Y_lag = Y_0

eld_del, eld_del_lag = 0.0, 0.0

guess3, guess4, guess5, guess9, guess10, guess13 = [1]*3,[1]*4, [1]*5, [1]*9,␣
↪[1]*10, [1]*13 # Arbitrary guesses for solving systems of equations

# Solve initial production values
init_production = fsolve(production_init, guess4)

# Pull computed output
Y_use = init_production[2]

# Solve FOCs for 2021
init_foc = fsolve(foc_init, guess5)

# Pull computed wage and cost of capital
w_use, r_use = init_foc[0], init_foc[1]

# Solve Income Identities for 2021, using FOCs and Production
init_ids = fsolve(identities_init, guess3)

# Pull computed government spending
G_use = init_ids[2]

# A few more initial conditions
```

```
EGHI_use = 0
SS_use = 1.144994521
GHC = 2.256568945
fed_med = medicaid_0*(1-medicaid_fed_share)
fed_other = other_0*(1-other_fed_share)

Bf_use = SS_use + GHC - fed_med - fed_other

# Solve initial Federal Government using FOCs and Income Identities
init_gov = fsolve(fed_gov_init, guess10)

# Store all computed values as row
initial = np.concatenate(([year], init_production, init_foc, init_ids, init_gov))

# Add the row to the 'projections' dataframe
projections.loc[len(projections)] = initial

# Now we prepare to solve every other year

# df_stuff stores many variables needed for computations, but not necessarily␣
 ↪outcomes of interest.
# Note, there is some overlap between 'df_stuff' and 'projections'
df_stuff = pd.DataFrame(columns=['Year', 'L','L_1', 'L_2', 'K', 'K_1', 'K_2',␣
 ↪'a', 'p', 'r_plus', 'Y', 'D', 'Df', 'EGHI', 'rel_p'])
df_stuff['Year'] = years

# Pull initial values from previously solved system of equations for 2021
df_stuff.loc[df_stuff['Year'] == min(years), 'p'] = init_foc[2]
df_stuff.loc[df_stuff['Year'] == min(years), 'r_plus'] = init_gov[8]
df_stuff.loc[df_stuff['Year'] == min(years), 'Y'] = init_production[2]
df_stuff.loc[df_stuff['Year'] == min(years), 'D'] = init_gov[7]
df_stuff.loc[df_stuff['Year'] == min(years), 'Df'] = init_gov[0]
df_stuff.loc[df_stuff['Year'] == min(years), 'EGHI'] = 0
df_stuff.loc[df_stuff['Year'] == min(years), 'rel_p'] = 1

# Prepare DataFrame 'df_hc' which will store annual HC expenditure estimates
df_hc = pd.DataFrame(columns=['Year', 'El Adj'])
df_hc['Year'] = years

# Make a column for each payer type
payers = df_hc_exp['payer'].unique()
for payer in payers:
    df_hc[payer] = pd.Series(dtype=float)

    # Make a column for storing 'Sum of Parts'
df_hc['Sum'] = [0]*len(years)
```

```
# Primary loop - within each iteration, a system of equations is solved to␣
 ↪compute projections for a given year
# Each iteration is its own year
for year in years:
    if year==min(years):
        continue # Initial year already computed - so skip

    # Indexes used to locate certain variables -- later on, can index by year,␣
 ↪which makes this redundant.
    index2 = int(year-min(years))
    index = index2 - 1

    # Compute f2_hat first using Population and Spending Matrix, as well as %␣
 ↪changes in price and income
    price_change = (projections.loc[projections['Year'] == year-1, 'p'][index] -␣
 ↪projections.loc[projections['Year'] == min(years), 'p'][0])/projections.
 ↪loc[projections['Year'] == min(years), 'p'][0]

    income_change = (projections.loc[projections['Year'] == year-1,␣
 ↪'w'][index]*labor_df.loc[labor_df['Year'] == year-1, 'Labor_Hours'][index] \
                     - projections.loc[projections['Year'] == min(years),␣
 ↪'w'][0]*labor_df.loc[labor_df['Year'] == min(years), 'Labor_Hours'][0])/ \
                     (projections.loc[projections['Year'] == min(years),␣
 ↪'w'][0]*labor_df.loc[labor_df['Year'] == min(years), 'Labor_Hours'][0])

    # Pull current income and price elasticities
    price_el = df_el.loc[df_el['Year'] == year, 'Price_Elasticity'][index2]
    income_el = df_el.loc[df_el['Year'] == year, 'Income_Elasticity'][index2]

    # Compute combined elasticity factor
    el_adj_factor = 1+price_change*price_el + income_change*income_el

    # Compute total spending by age group, sex, and payer
    df_hc_exp['temp'] =␣
 ↪round(df_hc_exp['cms_pred_per_capita'],0)*df_hc_exp[str(year)]/
 ↪1000000000000*el_adj_factor*phc_nhe_adj # Using temporary rounding

    # Sum spending by payer category
    sums = df_hc_exp.groupby('payer')['temp'].sum()

    # Pull values to subtract out non-federal components from GHC later on
    medicaid = sums.loc['Medicaid']
    other_hc = sums.loc['Other Payers and Programs']

    # Store health spending by payer group and compute their sum
    for payer in sums.keys():
```

```
        df_hc.loc[df_hc['Year'] == year, payer] = sums.loc[payer]
        if payer != 'Total': # Use sum of parts to compute total spending for a␣
↪year, omitting the 'Total' category
            df_hc.loc[df_hc['Year'] == year, 'Sum'] += df_hc.loc[df_hc['Year']␣
↪== year, payer]

    # Compute total Government Health Expenditures
    df_hc['GHC'] = df_hc['Medicaid'] + df_hc['Medicare'] + df_hc['Other Payers␣
↪and Programs']
    # Subtract out non-Federal Medicaid and 'Other' spending
    df_hc['Fed_GHC'] = df_hc['GHC'] - (1-medicaid_fed_share)*df_hc['Medicaid'] -␣
↪(1-other_fed_share)*df_hc['Other Payers and Programs']
    # Store this total demand for healthcare to be used later
    f2_hat = df_hc.loc[df_hc['Year'] == year, 'Sum']

    df_hc.loc[df_hc['Year'] == min(years), 'Sum'] = df_hc.loc[df_hc['Year'] ==␣
↪min(years)+1, 'Sum'][1] # Equalize first two years
    # Compute employer group health insurance
    df_hc['EGHI'] = 0.3*df_hc['Sum']
    # Compute change in EGHI
    df_hc['EGHI_Growth'] = df_hc['EGHI'].diff()

    # Pull last year's f2_hat
    f2_hat_lag = df_hc.loc[df_hc['Year'] == year-1, 'Sum']

    # Pull or compute values needed to solve production function and FOCs

    # Growth factors
    g_1 = g_df.loc[g_df['Year'] == year, 'g_1'][index2]
    g_2 = g_df.loc[g_df['Year'] == year, 'g_2'][index2]
    g_3 = g_df.loc[g_df['Year'] == year, 'g_3'][index2]

    # Labor
    df_stuff.loc[df_stuff['Year'] == year, 'L'] = labor_df.loc[labor_df['Year']␣
↪== year, 'Labor_Hours']
    df_stuff.loc[df_stuff['Year'] == year, 'L_2'] = f2_hat*beta2/g_3
    df_stuff.loc[df_stuff['Year'] == year, 'L_1'] = df_stuff.
↪loc[df_stuff['Year'] == year, 'L'] - df_stuff.loc[df_stuff['Year'] == year,␣
↪'L_2']

    # Capital
    df_stuff.loc[df_stuff['Year'] == year, 'K'] = projections.
↪loc[projections['Year'] == year-1, 'K'][index]*(1-b)+projections.
↪loc[projections['Year'] == year-1, 'I'][index]
    df_stuff.loc[df_stuff['Year'] == year, 'K_2'] = f2_hat*beta3
```

```
    df_stuff.loc[df_stuff['Year'] == year, 'K_1'] = df_stuff.
↪loc[df_stuff['Year'] == year, 'K'] - df_stuff.loc[df_stuff['Year'] == year,␣
↪'K_2']*g_2

    # Store as variables for solving FOCs and Production
    L = df_stuff.loc[df_stuff['Year'] == year, 'L'][index2]
    K_ = df_stuff.loc[df_stuff['Year'] == year, 'K'][index2]
    K_1 = df_stuff.loc[df_stuff['Year'] == year, 'K_1'][index2]
    K_2 = df_stuff.loc[df_stuff['Year'] == year, 'K_2'][index2]
    L_1 = df_stuff.loc[df_stuff['Year'] == year, 'L_1'][index2]
    L_2 =  df_stuff.loc[df_stuff['Year'] == year, 'L_2'][index2]

    # Compute FOCs first
    focs = fsolve(foc, guess5)

    # Pull relative price
    df_stuff.loc[df_stuff['Year'] == year, 'p'] = focs[2]

    # Pull wage and cost of capital
    w = focs[0]
    r = focs[1]

    # Compute change in relative price
    p = df_stuff.loc[df_stuff['Year'] == year, 'p'][index2]/df_stuff.
↪loc[df_stuff['Year'] == min(years), 'p'][0]

    # Solve production functions
    production = fsolve(productions, guess4)

    # Pull Output
    Y = production[2]
    Y_lag = df_stuff.loc[df_stuff['Year'] == year-1, 'Y'][index]
    Y_init = df_stuff.loc[df_stuff['Year'] == min(years), 'Y'][0]

    # Pull lagged debt and deficit
    D_lag = df_stuff.loc[df_stuff['Year'] == year-1, 'D'][index]
    Df_lag = df_stuff.loc[df_stuff['Year'] == year-1, 'Df'][index]

    # Pull current and lagged measure of elderly population for r_plus␣
↪computation
    eld_del = pop['Delta'][index2]
    eld_del_lag = pop['Delta'][index]
    plus_65 = pop['65_plus'][index2]

    # Pull hours worked per week for the year
    hrs_wk = labor_df['Hours_per_Week'][index2]
```

```
    # Compute defense and non-defense spending
    Def = def_share*Y
    NDef = non_def_share*pop['Total'][index2]

    # Pull lagged and initial market cost of capital
    r_plus_lag = df_stuff.loc[df_stuff['Year'] == year-1, 'r_plus'][index]
    r_plus_init = df_stuff.loc[df_stuff['Year'] == min(years), 'r_plus'][0]

    # Define EGHI, GHC and Fed GHC for computations
    EGHI = df_hc['EGHI_Growth'][index2]
    fed_ghc = df_hc['Fed_GHC'][index2]
    ghc = df_hc['GHC'][index2]

    # Solve Federal Government and Identities
    gov_ids = fsolve(fed_gov_ids, guess13)

    # Pull all projected values together and add them as a row to our final␣
↪'projections' dataframe
    current = np.concatenate(([year], production, focs, gov_ids))
    projections.loc[len(projections)] = current

    # Populate df_stuff for next iteration
    df_stuff.loc[df_stuff['Year'] == year, 'p'] = focs[2]
    df_stuff.loc[df_stuff['Year'] == year, 'r_plus'] = gov_ids[11]
    df_stuff.loc[df_stuff['Year'] == year, 'Y'] = production[2]
    df_stuff.loc[df_stuff['Year'] == year, 'D'] = gov_ids[10]
    df_stuff.loc[df_stuff['Year'] == year, 'Df'] = gov_ids[3]
    df_stuff.loc[df_stuff['Year'] == year, 'EGHI'] = EGHI
    df_stuff.loc[df_stuff['Year'] == year, 'rel_p'] = p
```