homework

April 21, 2020

0.1 Task 1

[1]:
```
import pandas as pd
```

0.1.1 Data downloaded from https://unstats.un.org/

[2]:
```
all_prices = pd.read_csv('tablebyExpenditure.csv')
```

[3]:
```
all_prices
```

[3]:

	Year	Final consumption expenditure	Household consumption expenditure	\
0	2000	5714456000	4279775000	
1	2001	6139227000	4604656000	
2	2002	6890075000	5156151000	
3	2003	7861399000	5855904000	
4	2004	9117940000	6917972000	
5	2005	10797364000	8364825000	
6	2006	13952825000	11004162000	
7	2007	17523601000	13542629000	
8	2008	18954758000	14138215000	
9	2009	15016472000	11413972000	
10	2010	14642762000	11331315000	
11	2011	16172785000	12466948000	
12	2012	17153734000	13339553000	
13	2013	18126757000	14106819000	
14	2014	18640281000	14494805000	
15	2015	19124731000	14709411000	
16	2016	19614233000	15088021000	
17	2017	20882970000	16030459000	
18	2018	22354522000	17169505000	
19	2000	11829611000	8203962000	
20	2001	12437613000	8637434000	
21	2002	13099967000	9142992000	
22	2003	13972493000	9882057000	
23	2004	15189874000	10968542000	
24	2005	16449828000	12087257000	
25	2006	19065557000	14447258000	
26	2007	20675744000	15917342000	

27	2008	19521830000	14655861000
28	2009	16674495000	12285466000
29	2010	16525765000	12502839000
30	2011	17092790000	12962027000
31	2012	17586170000	13437268000
32	2013	18395284000	14196265000
33	2014	18634629000	14346627000
34	2015	19124731000	14709411000
35	2016	19473811000	14930729000
36	2017	20084052000	15394679000
37	2018	20922697000	16044706000

	General government final consumption expenditure	Gross capital formation	\
0	1434681000	1679943000	
1	1534571000	2081545000	
2	1733924000	2343797000	
3	2005495000	2875868000	
4	2199968000	3635135000	
5	2432539000	4758541000	
6	2948663000	6676189000	
7	3980972000	9363153000	
8	4816543000	8584263000	
9	3602500000	4173542000	
10	3311447000	3588844000	
11	3705837000	5157164000	
12	3814181000	5744647000	
13	4019938000	5485150000	
14	4145476000	5495458000	
15	4415320000	5523199000	
16	4526212000	5212107000	
17	4852511000	5873046000	
18	5185017000	6850483000	
19	3625649000	3164011000	
20	3800179000	3909416000	
21	3956975000	4179796000	
22	4090436000	5211740000	
23	4221332000	6345756000	
24	4362571000	7063349000	
25	4618299000	8657392000	
26	4758402000	10416526000	
27	4865969000	8890598000	
28	4389029000	5241070000	
29	4022926000	4246632000	
30	4130763000	6192089000	
31	4148902000	6080420000	
32	4199019000	5674874000	
33	4288002000	5413660000	

34		4415320000	5523199000
35		4543082000	5593775000
36		4689373000	6252128000
37		4877991000	7073260000

	Gross fixed capital formation	Changes in inventories	\
0	1.715298e+09	-3.535500e+07	
1	2.025315e+09	5.623000e+07	
2	2.043171e+09	3.006260e+08	
3	2.362534e+09	5.133340e+08	
4	3.174243e+09	4.608920e+08	
5	4.237679e+09	5.208620e+08	
6	5.823499e+09	8.526900e+08	
7	8.218285e+09	1.144868e+09	
8	7.815381e+09	7.688820e+08	
9	4.223492e+09	-4.995000e+07	
10	3.446909e+09	1.419350e+08	
11	4.473615e+09	6.835490e+08	
12	5.540939e+09	2.037080e+08	
13	5.276839e+09	2.083110e+08	
14	5.339122e+09	1.563360e+08	
15	5.372034e+09	1.511650e+08	
16	4.899901e+09	3.122060e+08	
17	5.558724e+09	3.143220e+08	
18	6.555713e+09	2.947700e+08	
19	3.099780e+09	6.490095e+07	
20	3.698513e+09	2.077912e+08	
21	3.553976e+09	6.105047e+08	
22	3.939095e+09	1.238805e+09	
23	5.055873e+09	1.256457e+09	
24	6.112277e+09	9.283803e+08	
25	6.988366e+09	1.626140e+09	
26	8.551088e+09	1.818113e+09	
27	7.679887e+09	1.181741e+09	
28	5.024499e+09	2.143815e+08	
29	4.033189e+09	2.105320e+08	
30	4.992671e+09	1.168575e+09	
31	5.798962e+09	2.780125e+08	
32	5.455304e+09	2.176456e+08	
33	5.438023e+09	-1.919611e+07	
34	5.372034e+09	1.511650e+08	
35	4.933305e+09	6.452765e+08	
36	5.491908e+09	7.425784e+08	
37	6.360672e+09	6.970464e+08	

	Exports of goods and services	Imports of goods and services	GDP
0	2526565000	3073744000	6847219000

1	2839023000	3613531000	7446263000
2	3071650000	3922979000	8382543000
3	3450294000	4648513000	9539047000
4	4317994000	6036144000	11034925000
5	5870659000	7839888000	13586676000
6	6836172000	10371472000	17093713000
7	8686850000	12984090000	22589514000
8	9629045000	12774429000	24393637000
9	8020670000	8325785000	18884898000
10	9624988000	9889456000	17967137000
11	11738382000	12749027000	20319305000
12	13417954000	14391172000	21925163000
13	13741237000	14550131000	22803014000
14	14476868000	14958441000	23654165000
15	14831459000	15053431000	24425959000
16	15144263000	14897966000	25072637000
17	16648516000	16606699000	26797833000
18	17870755000	17924730000	29151030000
19	5262170000	6072490000	14190884000
20	5737748000	7022185000	15087617000
21	6026379000	7212551000	16161246000
22	6266238000	8068333000	17524671000
23	7126575000	9764953000	18986468000
24	8801555000	11411240000	21026256000
25	9460512000	13851926000	23525876000
26	10762024000	16255139000	25881018000
27	11017280000	14510717000	25014944000
28	9597688000	9908767000	21453280000
29	10887860000	11138641000	20493580000
30	12199384000	13592628000	21781629000
31	13392377000	14324068000	22682131000
32	13536684000	14376500000	23210173000
33	14412484000	14804760000	23654691000
34	14831459000	15053431000	24425959000
35	15419186000	15627544000	24859229000
36	16403633000	16933803000	25800726000
37	17052170000	18018269000	26992920000

0.1.2 Normalizing factors for prices (billions) and populations (millions)

[4]:
```
mult_prices = 1000000000
mult_population = 1000000
```

0.1.3 Resulting data tables

[5]:
```
constant_prices = all_prices.iloc[19:]
constant_prices.rename(columns={'Household consumption expenditure': 'H',
                                'Gross capital formation': 'G',
                                'GDP': 'Y'}, inplace=True)
constant_prices = constant_prices[['Year', 'H', 'G', 'Y']].reset_index()
constant_prices[['H','G', 'Y']] /= mult_prices
constant_prices
```

```
/home/elavrukhin/.local/lib/python3.6/site-packages/pandas/core/frame.py:4133:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

[5]:

	index	Year	H	G	Y
0	19	2000	8.203962	3.164011	14.190884
1	20	2001	8.637434	3.909416	15.087617
2	21	2002	9.142992	4.179796	16.161246
3	22	2003	9.882057	5.211740	17.524671
4	23	2004	10.968542	6.345756	18.986468
5	24	2005	12.087257	7.063349	21.026256
6	25	2006	14.447258	8.657392	23.525876
7	26	2007	15.917342	10.416526	25.881018
8	27	2008	14.655861	8.890598	25.014944
9	28	2009	12.285466	5.241070	21.453280
10	29	2010	12.502839	4.246632	20.493580
11	30	2011	12.962027	6.192089	21.781629
12	31	2012	13.437268	6.080420	22.682131
13	32	2013	14.196265	5.674874	23.210173
14	33	2014	14.346627	5.413660	23.654691
15	34	2015	14.709411	5.523199	24.425959
16	35	2016	14.930729	5.593775	24.859229
17	36	2017	15.394679	6.252128	25.800726
18	37	2018	16.044706	7.073260	26.992920

[6]:
```
current_prices = all_prices.iloc[:19]
current_prices.rename(columns={'Household consumption expenditure': 'H',
                               'Gross capital formation': 'G',
                               'GDP': 'Y'}, inplace=True)
current_prices = current_prices[['Year', 'H', 'G', 'Y']].reset_index()
current_prices[['H','G', 'Y']] /= mult_prices
current_prices
```

```
/home/elavrukhin/.local/lib/python3.6/site-packages/pandas/core/frame.py:4133:
SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

[6]:

	index	Year	H	G	Y
0	0	2000	4.279775	1.679943	6.847219
1	1	2001	4.604656	2.081545	7.446263
2	2	2002	5.156151	2.343797	8.382543
3	3	2003	5.855904	2.875868	9.539047
4	4	2004	6.917972	3.635135	11.034925
5	5	2005	8.364825	4.758541	13.586676
6	6	2006	11.004162	6.676189	17.093713
7	7	2007	13.542629	9.363153	22.589514
8	8	2008	14.138215	8.584263	24.393637
9	9	2009	11.413972	4.173542	18.884898
10	10	2010	11.331315	3.588844	17.967137
11	11	2011	12.466948	5.157164	20.319305
12	12	2012	13.339553	5.744647	21.925163
13	13	2013	14.106819	5.485150	22.803014
14	14	2014	14.494805	5.495458	23.654165
15	15	2015	14.709411	5.523199	24.425959
16	16	2016	15.088021	5.212107	25.072637
17	17	2017	16.030459	5.873046	26.797833
18	18	2018	17.169505	6.850483	29.151030

[7]:
```
population = pd.read_csv('tableExPop.csv').rename(columns={'P_Value': 'N'})
population = population[['Year', 'N']]
population['N'] /= mult_population
population
```

[7]:

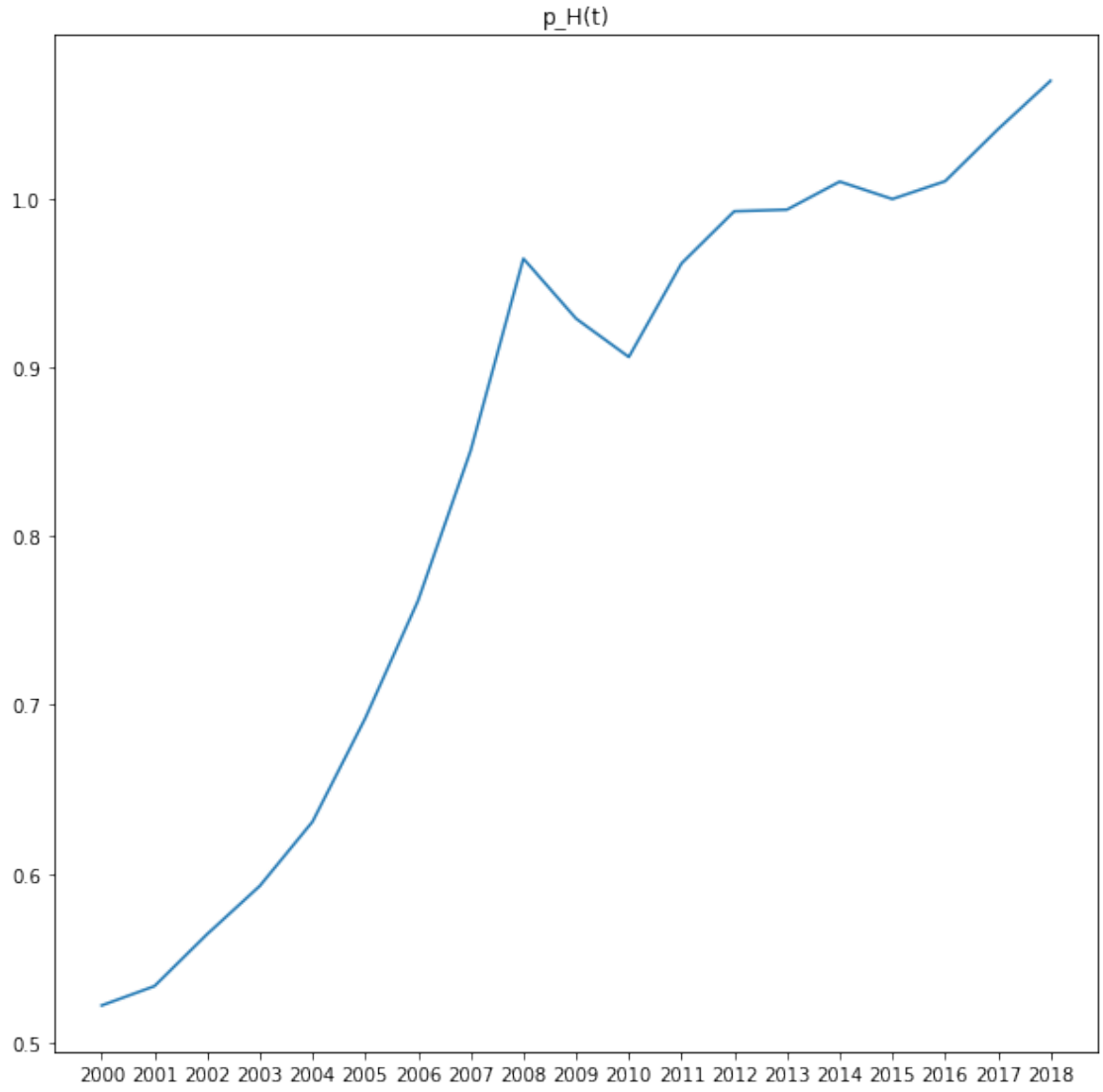
	Year	N
0	2000	2.384164
1	2001	2.358693
2	2002	2.332530
3	2003	2.305848
4	2004	2.278921
5	2005	2.251993
6	2006	2.225066
7	2007	2.198089
8	2008	2.171259
9	2009	2.144785
10	2010	2.118861
11	2011	2.093610
12	2012	2.069016
13	2013	2.044957
14	2014	2.021219

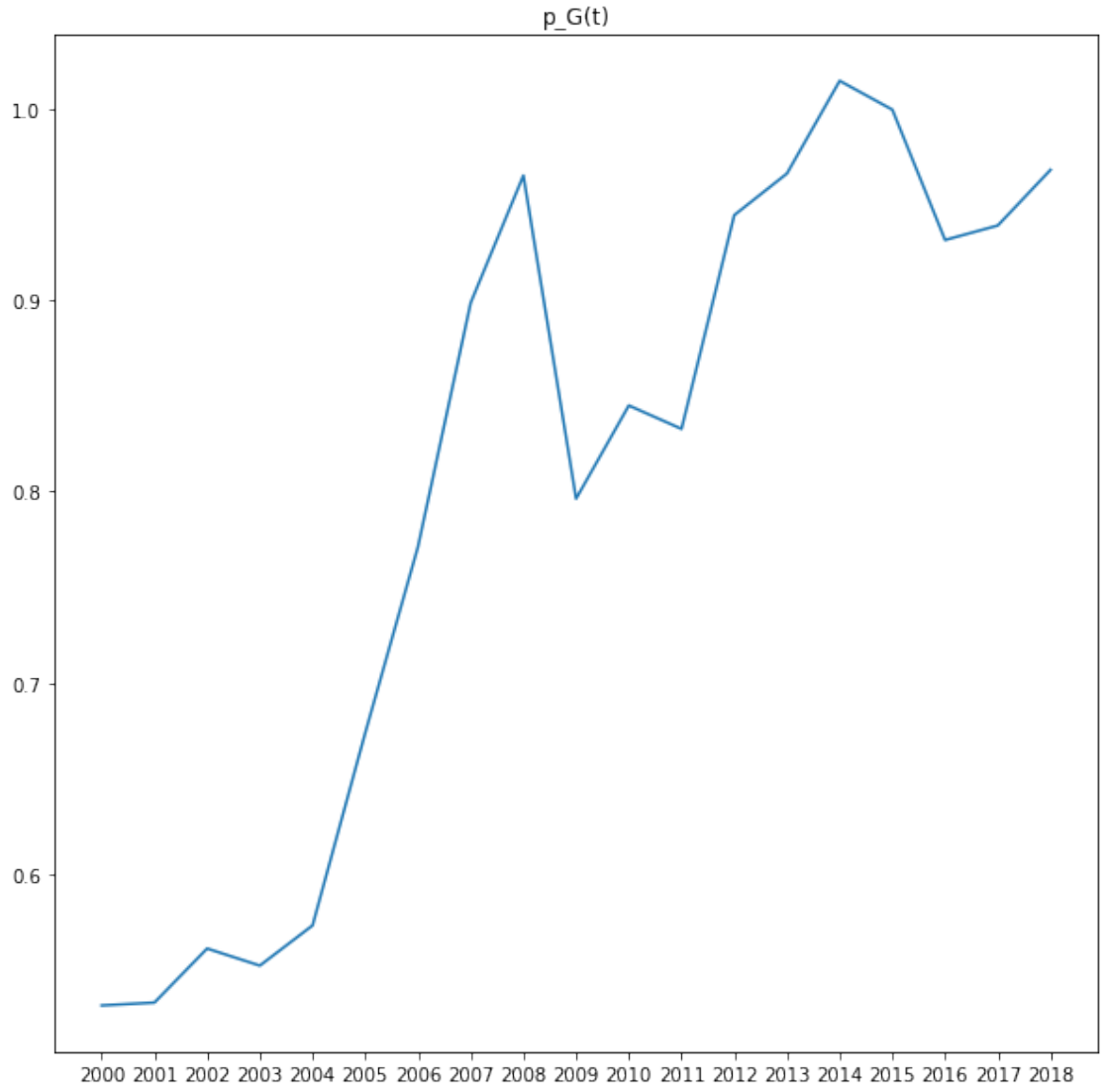
```
15  2015  1.997674
16  2016  1.974266
17  2017  1.951097
18  2018  1.928459
```

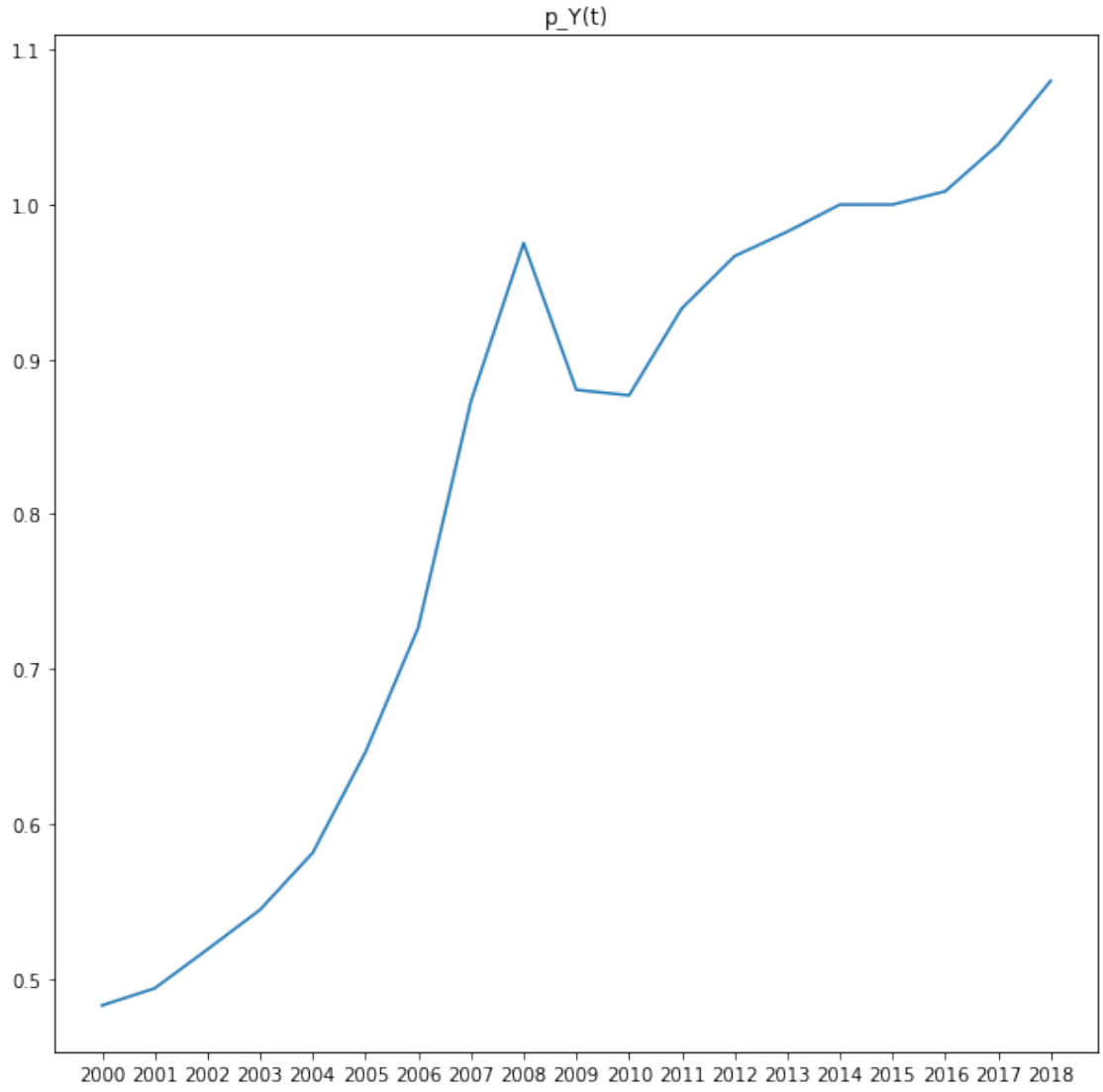
0.1.4 Household consumption expenditure, Gross capital formation and Gross Domestic Product graphics

[8]:
```
import matplotlib.pyplot as plt
```

[9]:
```
for x_name in ['H', 'G', 'Y']:
    x = current_prices[x_name] / constant_prices[x_name]
    dates = [str(t) for t in current_prices['Year']]
    plt.figure(figsize=(10, 10))
    plt.plot(dates, x)
    plt.title('p_{}(t)'.format(x_name))
    plt.show()
```


p_H(t)
1.0
0.9
0.8
0.7
0.6
0.5
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018


p_G(t)
1.0
0.9
0.8
0.7
0.6
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018



0.2 Task 2

0.2.1 Data downloaded from link below:

[10]:
```
url = 'https://data.worldbank.org/indicator/SL.EMP.TOTL.SP.ZS?
↪end=2018&locations=LV&start=2000&view=chart'
```

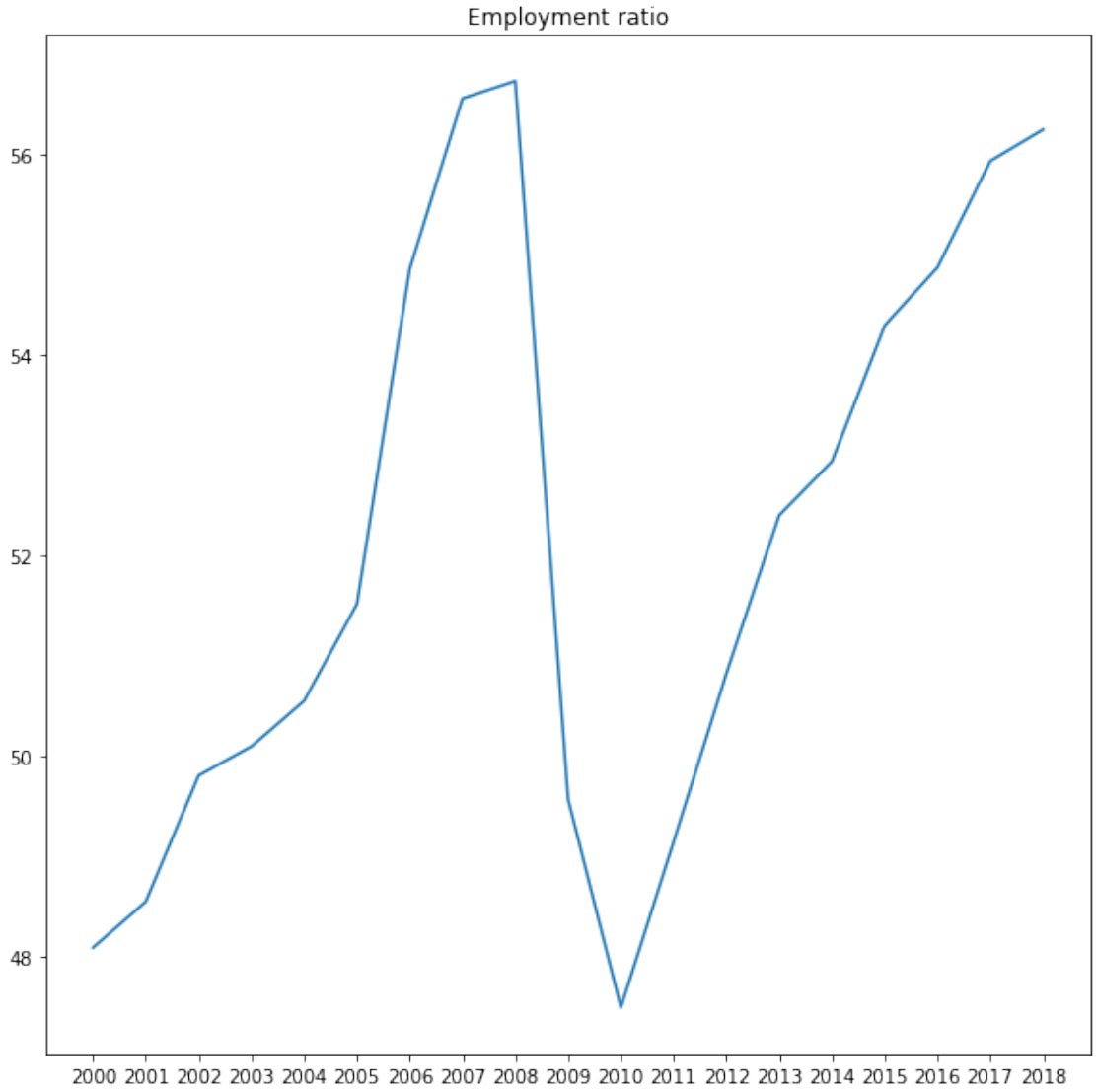
[11]:
```
employment_data = '48.0880012512207        48.5470008850098        49.
↪8050003051758        50.0940017700195        50.5499992370605        51.
↪5169982910156        54.859001159668        56.5589981079102        56.
↪7330017089844        49.5670013427734        47.4930000305176        49.
↪1440010070801        50.8250007629395        52.4020004272461        52.
↪9379997253418        54.2949981689453        54.875        55.
↪9360008239746        56.2490005493164        55.7480010986328'
```

```
employment_data = [float(item) for item in employment_data.split()][:-1]
employment_data
```

[11]:
```
[48.0880012512207,
 48.5470008850098,
 49.8050003051758,
 50.0940017700195,
 50.5499992370605,
 51.5169982910156,
 54.859001159668,
 56.5589981079102,
 56.7330017089844,
 49.5670013427734,
 47.4930000305176,
 49.1440010070801,
 50.8250007629395,
 52.4020004272461,
 52.9379997253418,
 54.2949981689453,
 54.875,
 55.9360008239746,
 56.2490005493164]
```

0.2.2 Employment ratio graphic

[12]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, employment_data)
plt.title('Employment ratio')
plt.show()
```



0.3 Task 3

[13]:
```
import numpy as np
```

[14]:
```
Y_stat = np.array(constant_prices['Y'])
J_stat = np.array(constant_prices['G'])
L_stat = np.array(employment_data) / 100
```

0.3.1 Selected parameters values

[15]:
```
b = 3.9387755102040813
eps = 0.44
mu = 0.001
nu0 = 0.017881569011991897
```

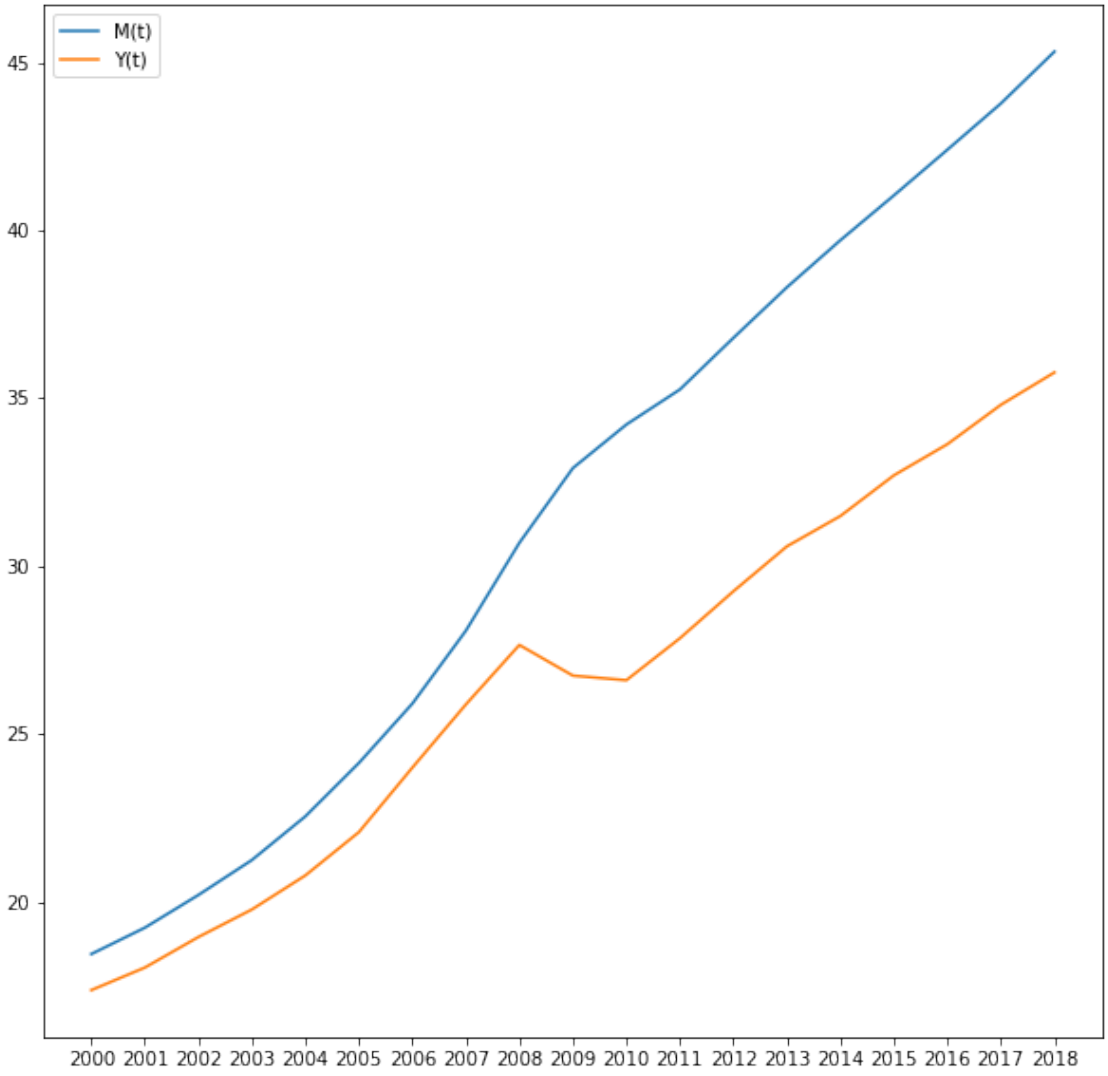
0.3.2 Model calculations

[16]:
```
M = np.zeros_like(Y_stat)
M[0] = 1.3 * Y_stat[0]
for t in range(0, Y_stat.shape[0] - 1):
    M[t + 1] = M[t] * (1 - mu) + J_stat[t] / b
M

x = np.zeros_like(L_stat)
x = L_stat / M
x

sigma = J_stat / (b * M)
sigma

nu = np.zeros_like(Y_stat)
nu[0] = nu0
for t in range(0, Y_stat.shape[0] - 1):
    nu[t+1] = nu[t] * (1 - eps * sigma[t])
nu

def f(x, alpha, mu):
    return (1 - (1 - alpha * x / nu) ** (1 / alpha))

alpha = 1 - eps - mu / sigma
Y = M * f(x, alpha, nu)
Y
```

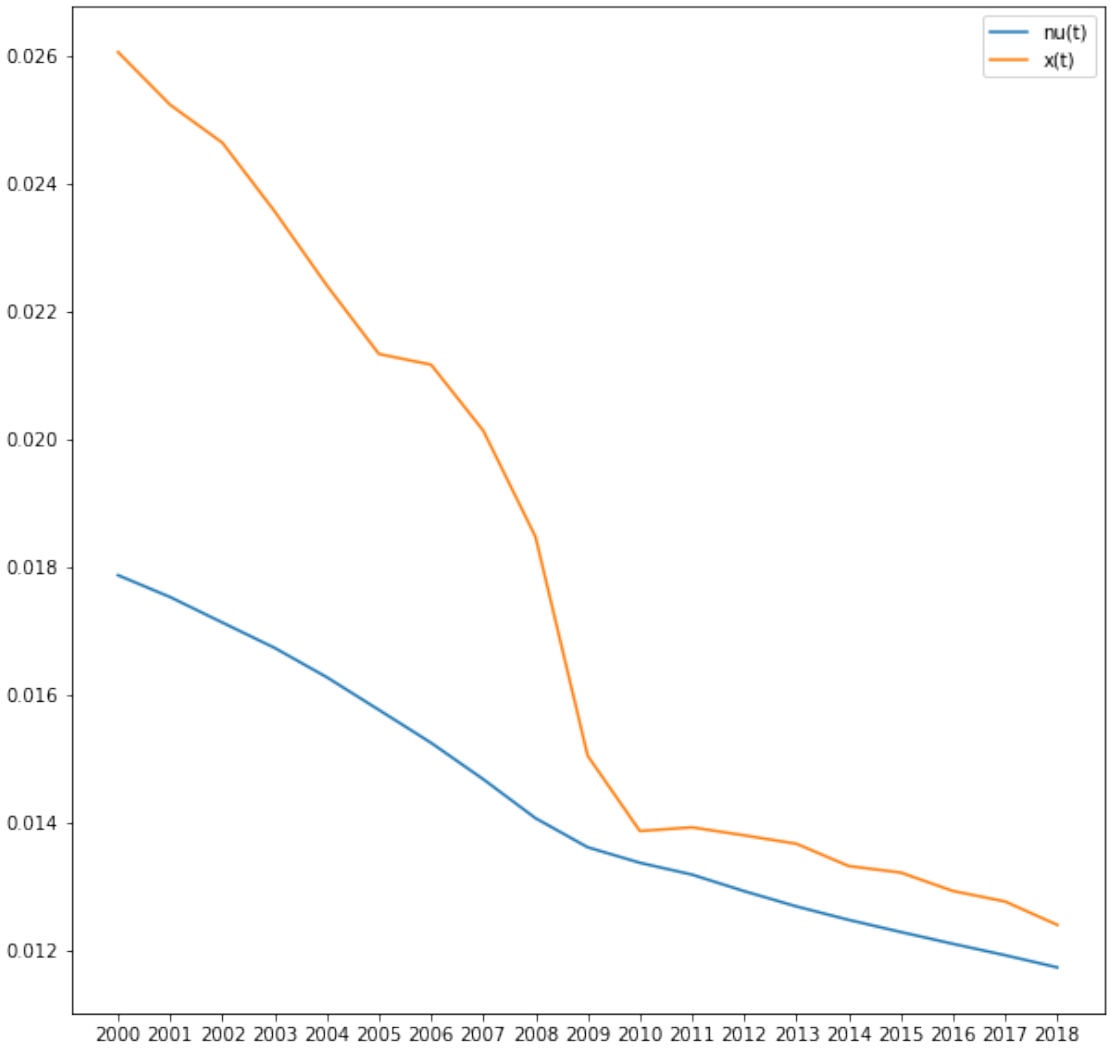
[16]:
```
array([18.4481492 , 19.23299918, 20.20631221, 21.24729763, 22.54923821,
       24.13778764, 25.90693535, 28.07901913, 30.69555034, 32.92205325,
       34.21976555, 35.26370624, 36.8005273 , 38.30746035, 39.70992401,
       41.04466662, 42.40588492, 43.78366025, 45.32720442])
```

0.3.3 Calculated variables graphics

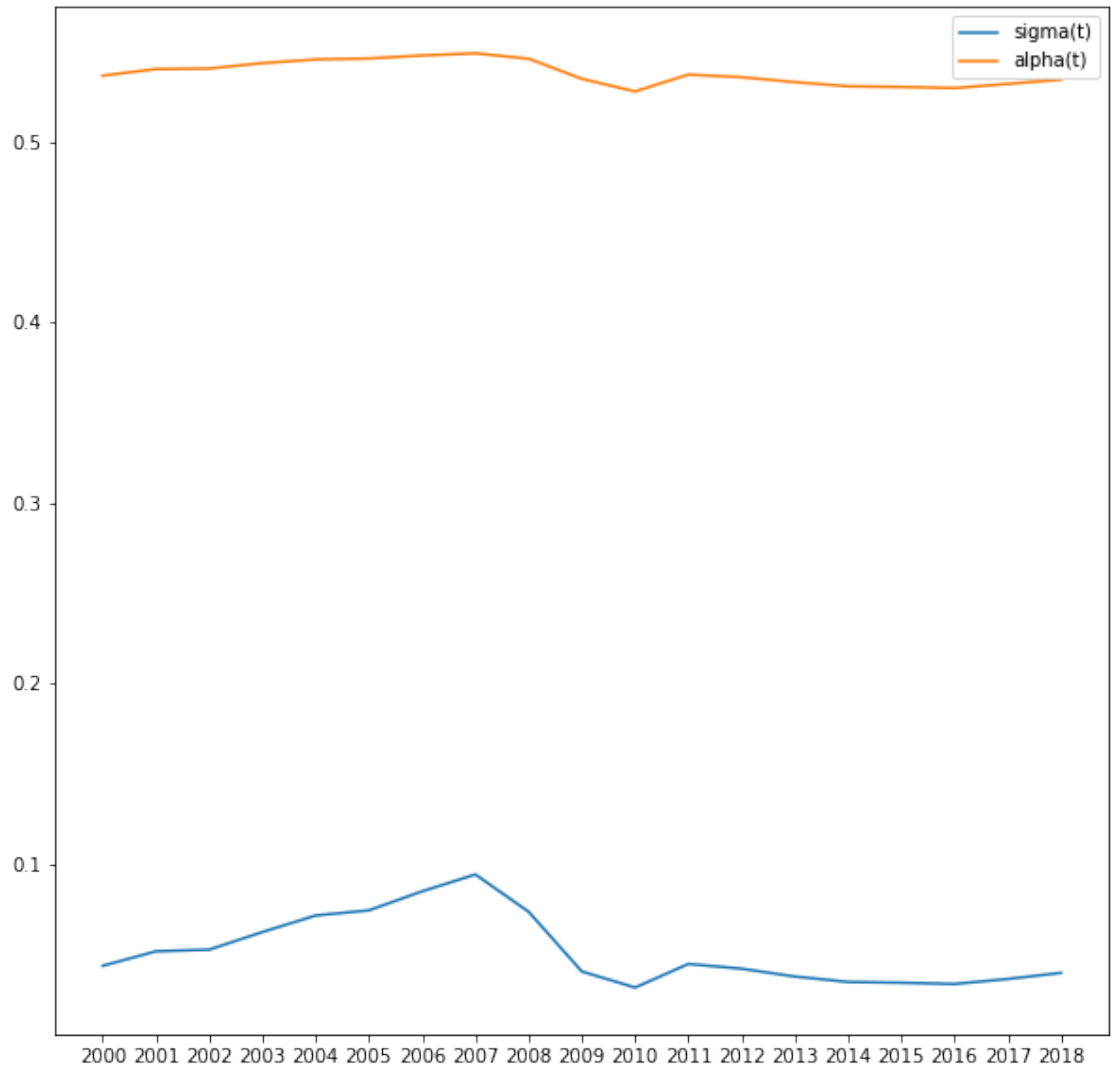
[22]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, M, label='M(t)')
plt.plot(dates, Y, label='Y(t)')
plt.legend(loc='best')
plt.show()
```



[23]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, nu, label='nu(t)')
plt.plot(dates, x, label='x(t)')
plt.legend(loc='best')
plt.show()
```



[24]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, sigma, label='sigma(t)')
plt.plot(dates, alpha, label='alpha(t)')
plt.legend(loc='best')
plt.show()
```



0.3.4 Grid search for model parameters

[25]:
```
from tqdm import tqdm
```

[ ]:
```
results = []
for i, b in tqdm(enumerate(np.linspace(1, 4, 50)), total=50):
    for eps in np.linspace(0.01, 0.5, 50):
        for mu in np.linspace(0.001, 0.05, 50):
            for nu0 in np.linspace(0.001, L_stat[0] / (1.3 * Y_stat[0]), 50):
                M = np.zeros_like(Y_stat)
                M[0] = 1.3 * Y_stat[0]
                for t in range(0, Y_stat.shape[0] - 1):
                    M[t + 1] = M[t] * (1 - mu) + J_stat[t] / b
                x = np.zeros_like(L_stat)
```

```
                x = L_stat / M
                sigma = J_stat / (b * M)
                nu = np.zeros_like(Y_stat)
                nu[0] = nu0
                for t in range(0, Y_stat.shape[0] - 1):
                    nu[t+1] = nu[t] * (1 - eps * sigma[t])
                alpha = 1 - eps - mu / sigma
                Y = M * f(x, alpha, nu)

                success = (
                    np.all(nu < x)
                    and np.all(sigma > 0)
                    and np.all(sigma < 0.15)
                    and np.all((1 - alpha * x / nu) > 0)
                )
                if success:
                    results.append({'b': b, 'eps': eps, 'mu': mu, 'nu0': nu0})
    if results:
        print('Iteration {}: {} points'.format(i, len(results)))
```

0.3.5 Searched intervals for parameters

[29]:
```
b = set([item['b'] for item in results])
eps = set([item['eps'] for item in results])
mu = set([item['mu'] for item in results])
nu0 = set([item['nu0'] for item in results])
```

[30]:
```
print('b in [{}, {}]'.format(min(b), max(b)))
print('eps in [{}, {}]'.format(min(eps), max(eps)))
print('mu in [{}, {}]'.format(min(mu), max(mu)))
print('nu0 = [{}, {}]'.format(min(nu0), max(nu0)))
```

```
b in [1.7346938775510203, 4.0]
eps in [0.01, 0.5]
mu in [0.001, 0.05]
nu0 = [0.003046250789332351, 0.025555009471988216]
```

0.4 Task 4

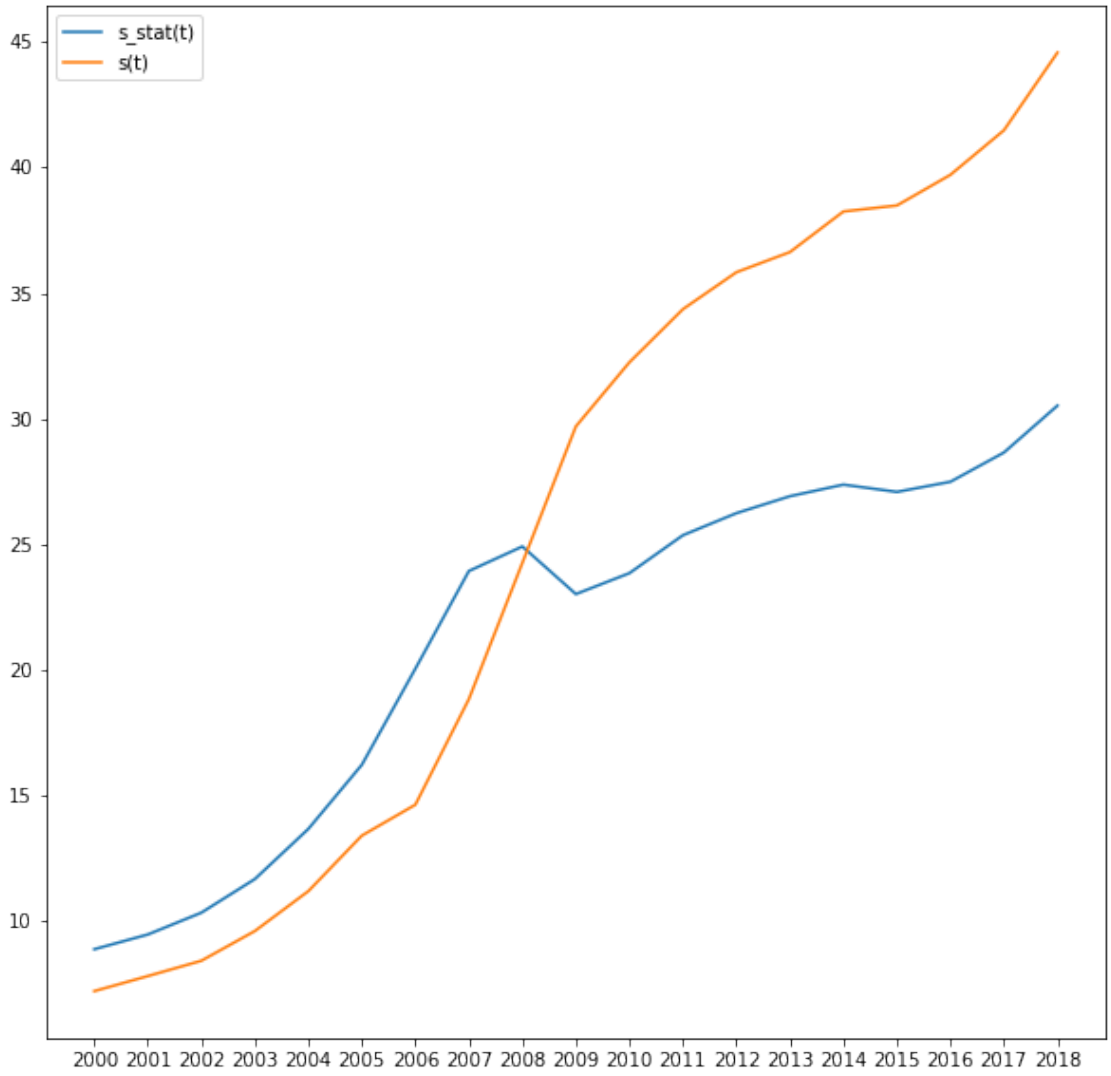
[26]:
```
p_stat = current_prices['Y'] / constant_prices['Y']
```

[27]:
```
s_stat = current_prices['H'] / L_stat
```

[28]:
```
s = p_stat / nu * (1 - alpha * x / nu) ** ((1 - alpha) / alpha)
```

[29]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, s_stat, label='s_stat(t)')
plt.plot(dates, s, label='s(t)')
```

```
plt.legend(loc='best')
plt.show()
```



0.5 Task 5

0.5.1 Search for parameters with a minimum value of the Theil index

[ ]:
```
results = []
for i, b in tqdm(enumerate(np.linspace(1, 4, 100)), total=100):
    for eps in np.linspace(0.01, 0.5, 100):
        for mu in np.linspace(0.001, 0.05, 100):
            for nu0 in np.linspace(0.001, L_stat[0] / (1.3 * Y_stat[0]), 100):
                M = np.zeros_like(Y_stat)
                M[0] = 1.3 * Y_stat[0]
```

```
                for t in range(0, Y_stat.shape[0] - 1):
                    M[t + 1] = M[t] * (1 - mu) + J_stat[t] / b
                x = np.zeros_like(L_stat)
                x = L_stat / M
                sigma = J_stat / (b * M)
                nu = np.zeros_like(Y_stat)
                nu[0] = nu0
                for t in range(0, Y_stat.shape[0] - 1):
                    nu[t+1] = nu[t] * (1 - eps * sigma[t])
                alpha = 1 - eps - mu / sigma
                Y = M * f(x, alpha, nu)

                success = (
                    np.all(nu < x)
                    and np.all(sigma > 0)
                    and np.all(sigma < 0.15)
                    and np.all((1 - alpha * x / nu) > 0)
                )
                if success:
                    s = p_stat / nu * (1 - alpha * x / nu) ** ((1 - alpha) /␣
↪alpha)
                    T = np.sqrt(np.square(s - s_stat).sum() / (np.square(s) +␣
↪np.square(s_stat)).sum())
                    results.append({'T': T, 'b': b, 'eps': eps, 'mu': mu, 'nu0':
↪ nu0})
    if results:
        print('Iteration {}: {} points'.format(i, len(results)))
```

[45]:
```
T_min = 1
idx = None
for i, item in enumerate(results):
    if item['T'] < T_min:
        idx = i
        T_min = item['T']
```

0.5.2 Minimal Theil index value

[56]:
```
print('Theil = {}'.format(results[idx]['T']))
```

```
Theil = 0.0724141136957671
```

0.5.3 Parameters from minimal Theil index value

[59]:
```
print('b = {}'.format(results[idx]['b']))
print('eps = {}'.format(results[idx]['eps']))
print('mu = {}'.format(results[idx]['mu']))
print('nu0 = {}'.format(results[idx]['nu0']))
```

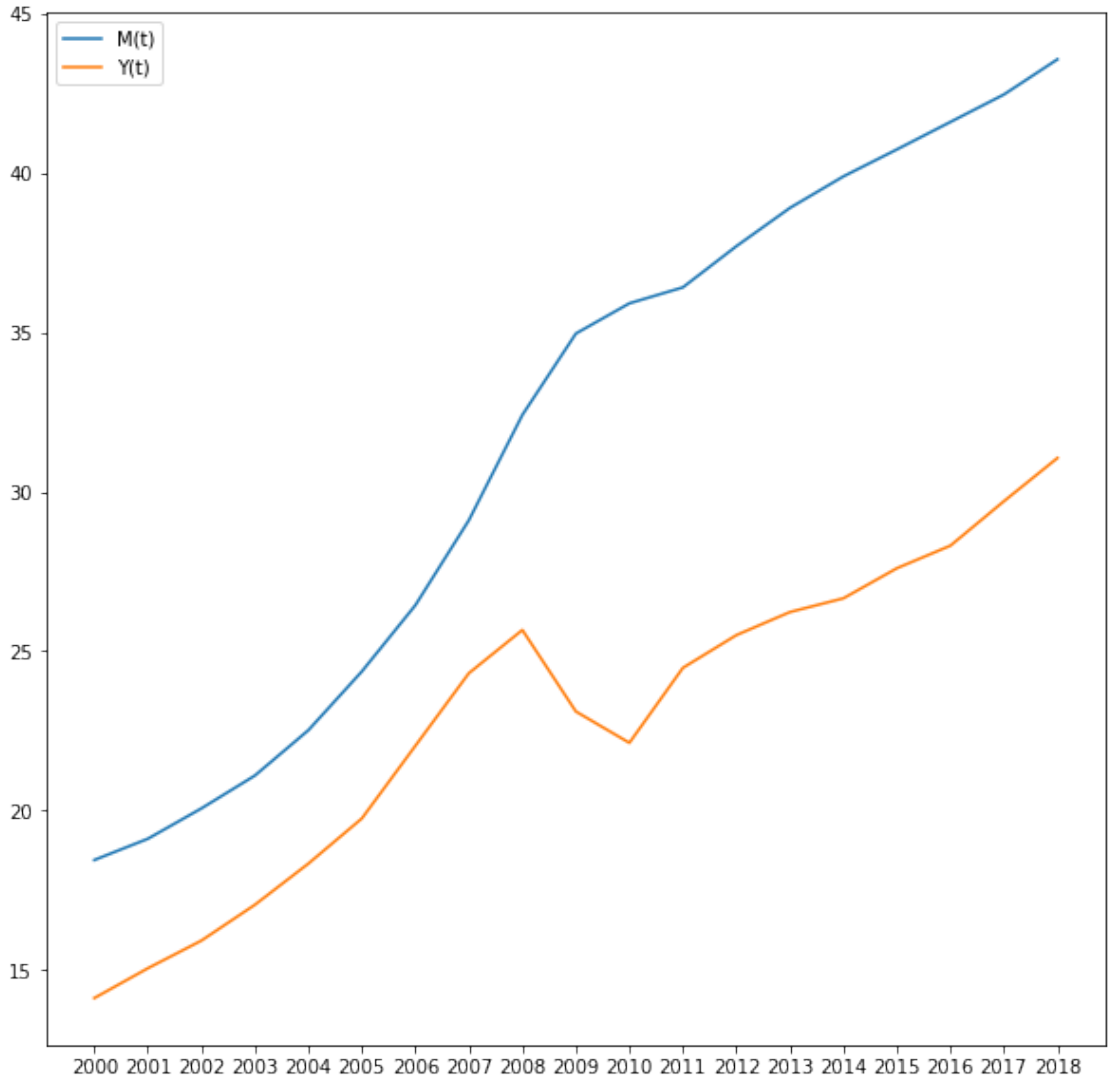
```
b = 2.4081632653061225
eps = 0.37626262626262624
mu = 0.035151515151515156
nu0 = 0.019736629702320974
```

[48]:
```
b = results[idx]['b']
eps = results[idx]['eps']
mu = results[idx]['mu']
nu0 = results[idx]['nu0']
```

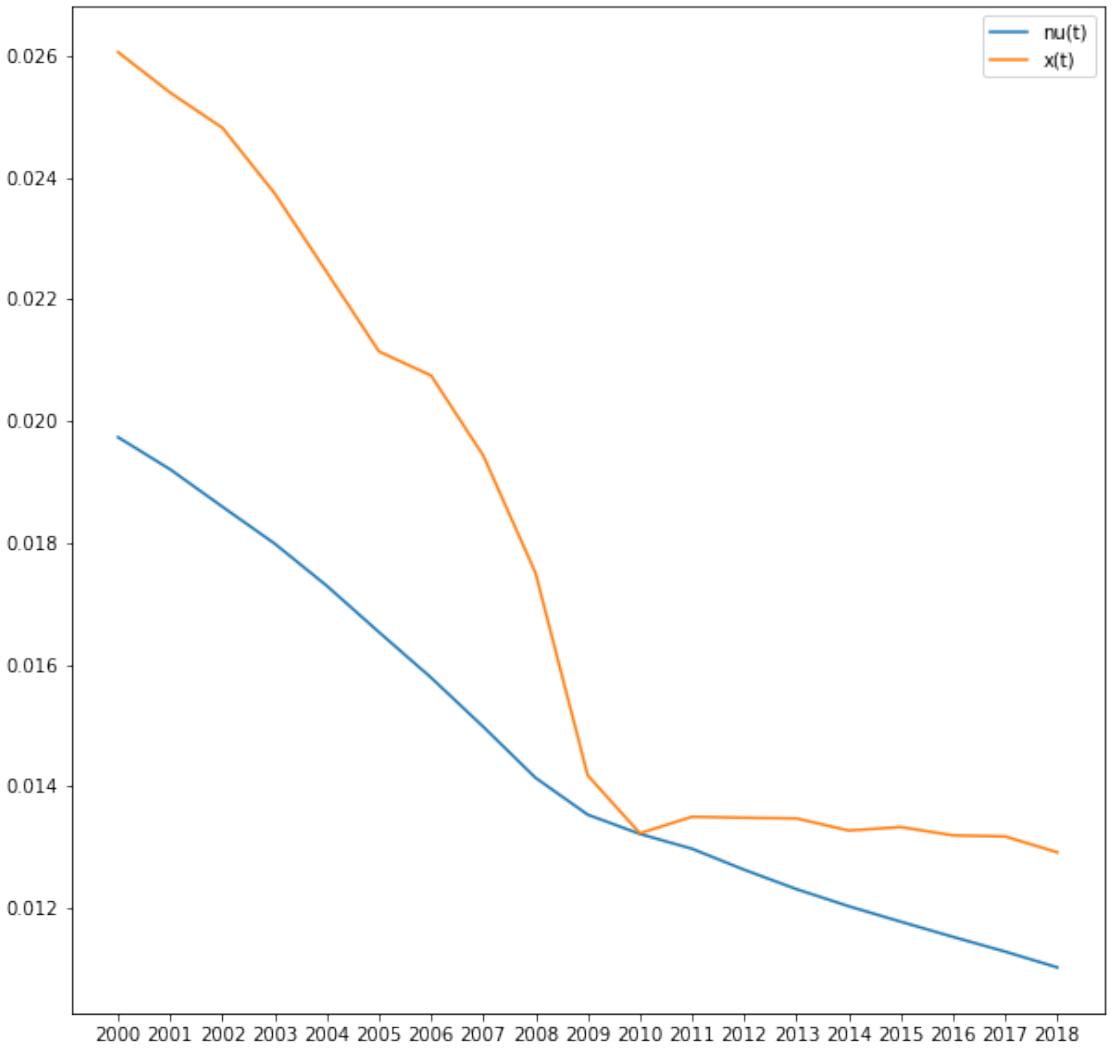
[54]:
```
M = np.zeros_like(Y_stat)
M[0] = 1.3 * Y_stat[0]
for t in range(0, Y_stat.shape[0] - 1):
    M[t + 1] = M[t] * (1 - mu) + J_stat[t] / b

x = np.zeros_like(L_stat)
x = L_stat / M

sigma = J_stat / (b * M)

nu = np.zeros_like(Y_stat)
nu[0] = nu0
for t in range(0, Y_stat.shape[0] - 1):
    nu[t+1] = nu[t] * (1 - eps * sigma[t])

def f(x, alpha, mu):
    return (1 - (1 - alpha * x / nu) ** (1 / alpha))

alpha = 1 - eps - mu / sigma
Y = M * f(x, alpha, nu)
```

0.5.4 Graphs of all indicators for the desired parameter values

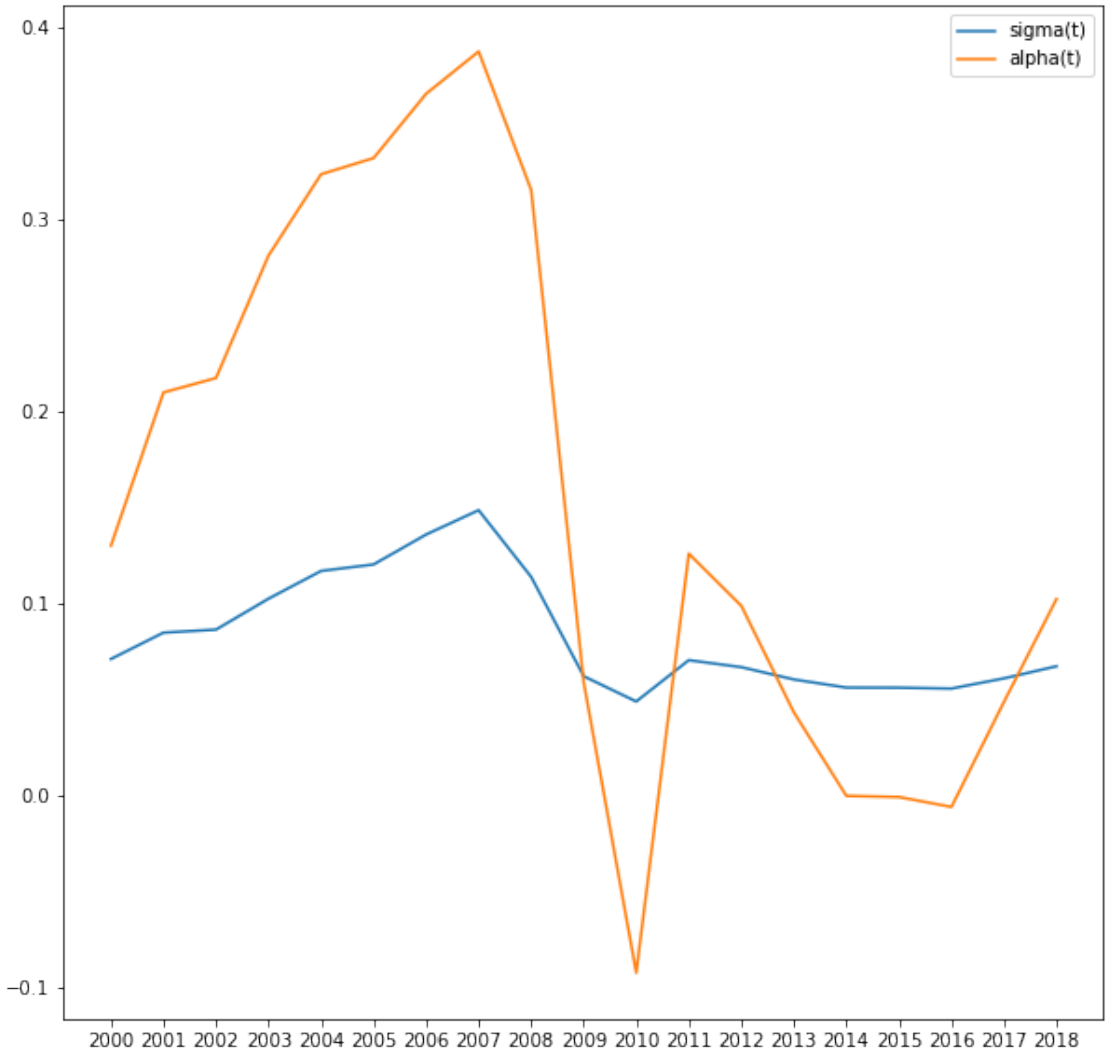
[50]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, M, label='M(t)')
plt.plot(dates, Y, label='Y(t)')
plt.legend(loc='best')
plt.show()
```



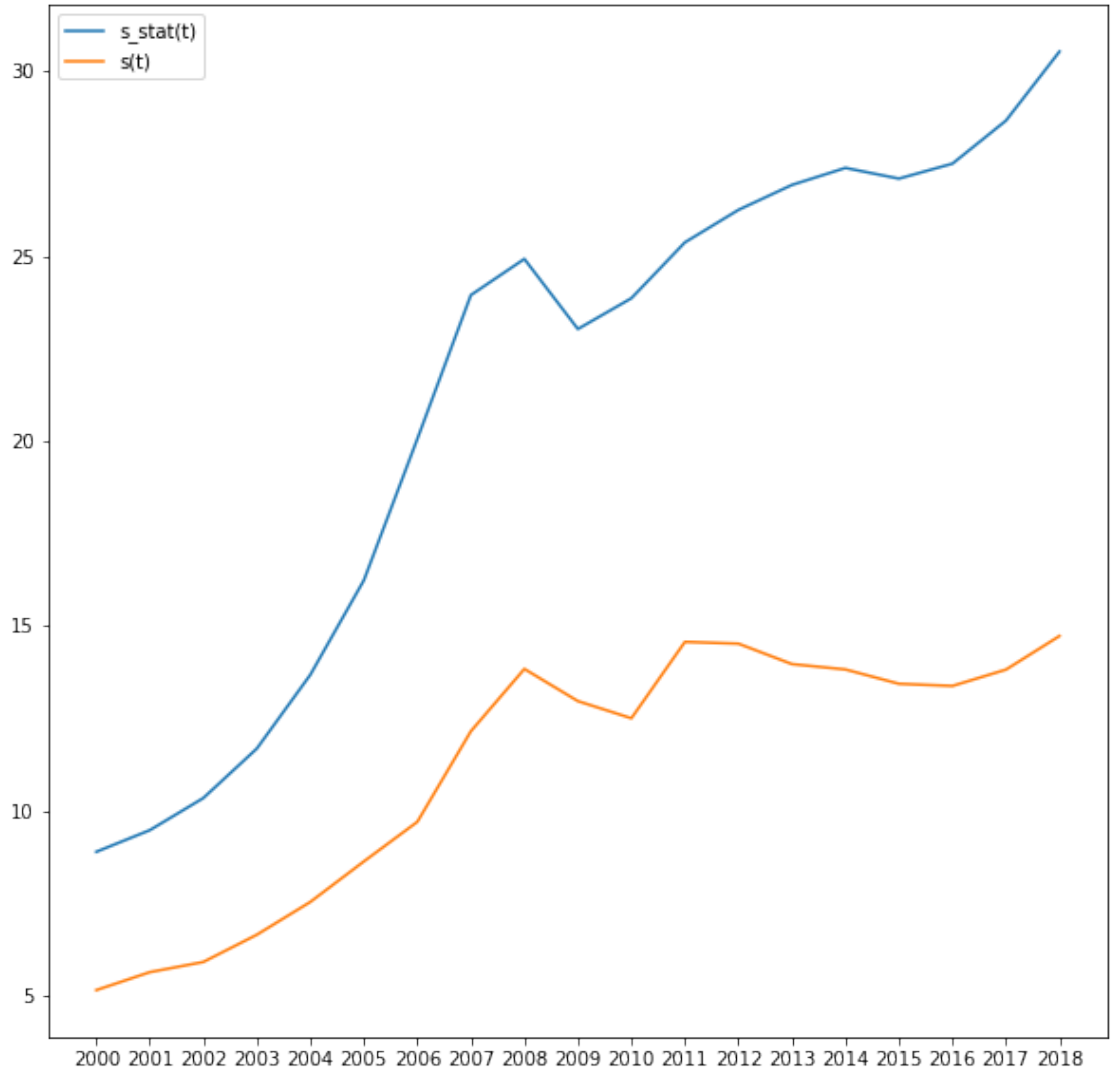
[51]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, nu, label='nu(t)')
plt.plot(dates, x, label='x(t)')
plt.legend(loc='best')
plt.show()
```



[52]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, sigma, label='sigma(t)')
plt.plot(dates, alpha, label='alpha(t)')
plt.legend(loc='best')
plt.show()
```



[53]:
```
plt.figure(figsize=(10, 10))
plt.plot(dates, s_stat, label='s_stat(t)')
plt.plot(dates, s, label='s(t)')
plt.legend(loc='best')
plt.show()
```


s_stat(t)
s(t)
30
25
20
15
10
5
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018