In [90]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
from scipy.stats import norm
import copy
import pandas as pd
```

Q1 (a)

In [91]:
```
N = 9
m = 3
mu = 3
beta = 0.96
phi = 1
rho = 0.3
sigma_sq_er = 0.01 * (1-(rho)**2)
sigma_sq_z = sigma_sq_er/(1-rho**2)
zn = m*(sigma_sq_z)**(1/2)
z0 = -zn
s = (zn-z0)/(N-1)
z_grid = np.arange(z0,zn+s,s)
print(z_grid)
b = np.zeros((9,9))
for j in range(N):
    for i in range(N):
        if j==0:
            b[i,j] = norm.cdf((z0-rho*z_grid[i]+s/2)/sigma_sq_er**(1/2))
        elif j==N-1:
            b[i,j] = 1-norm.cdf((zn-rho*z_grid[i]-s/2)/sigma_sq_er**(1/2))
        else:
            b[i,j] = norm.cdf((z_grid[j] - rho*z_grid[i] + s/2)/sigma_sq_er**(1/2))



print(b)
print(np.exp(z_grid))
```

```
[-0.3   -0.225 -0.15  -0.075  0.     0.075  0.15   0.225  0.3  ]
[[3.52805842e-02 1.18091475e-01 2.53397008e-01 3.02191106e-01
  2.00357192e-01 7.37963542e-02 1.50732903e-02 1.70313471e-03
  1.09855133e-04]
 [2.04690973e-02 8.37373046e-02 2.14354009e-01 3.04863406e-01
  2.41062359e-01 1.05927790e-01 2.58286367e-02 3.48664672e-03
  2.70750483e-04]
 [1.13032654e-02 5.63099151e-02 1.71986068e-01 2.91733854e-01
  2.75102216e-01 1.44200358e-01 4.19648527e-02 6.76624640e-03
  6.33224834e-04]
 [5.93671933e-03 3.59079082e-02 1.30879138e-01 2.64803013e-01
  2.97789168e-01 1.86176853e-01 6.46532749e-02 1.24480924e-02
  1.40583226e-03]
 [2.96390859e-03 2.17123197e-02 9.44590725e-02 2.27984995e-01
  3.05759408e-01 2.27984995e-01 9.44590725e-02 2.17123197e-02
  2.96390859e-03]
 [1.40583226e-03 1.24480924e-02 6.46532749e-02 1.86176853e-01
  2.97789168e-01 2.64803013e-01 1.30879138e-01 3.59079082e-02
  5.93671933e-03]
 [6.33224834e-04 6.76624640e-03 4.19648527e-02 1.44200358e-01
  2.75102216e-01 2.91733854e-01 1.71986068e-01 5.63099151e-02
  1.13032654e-02]
```

```
 [2.70750483e-04 3.48664672e-03 2.58286367e-02 1.05927790e-01
  2.41062359e-01 3.04863406e-01 2.14354009e-01 8.37373046e-02
  2.04690973e-02]
 [1.09855133e-04 1.70313471e-03 1.50732903e-02 7.37963542e-02
  2.00357192e-01 3.02191106e-01 2.53397008e-01 1.18091475e-01
  3.52805842e-02]]
[0.74081822 0.79851622 0.86070798 0.92774349 1.         1.07788415
 1.16183424 1.25232272 1.34985881]
```

Q1 (b):

Primeira parte é uma tentativa mais elegante, o segundo bloco é mais 'braçal'

In [92]:

```
a_size=201
valor = np.zeros((1,a_size))
print(valor[:,1])
```

```
[0.]
```

In [93]:

```
# ORDEM DA FUNÇÃO VALOR: LINHA É a, COLUNA É Z
a_size=201
a_max=4
tol = 10**(-6)
r=0.04
V_0 = np.zeros((a_size, N))
grid_a = np.linspace(-phi, a_max, a_size)
print(grid_a)
V_new = copy.deepcopy(V_0)
erro = 1
g = np.zeros((a_size, N))
t= 1
while(erro > tol):
    V_old = copy.deepcopy(V_new)
    t += 1
    for i in range(N):
        probabilidades = b[i,:]
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = 0
            valor = np.zeros((1,a_size))
            t = copy.deepcopy(start)
            c_t = (estoque - grid_a[t])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca = 0
            for w in range(N):
                esperanca = (esperanca + probabilidades[w]*V_old[t,w])
            valor[:,t] = (u + beta*esperanca)
            for t in range((start+1), a_size):
                c_t = (estoque - grid_a[t])
                if c_t >= 0:
                    u = ((c_t**(1-mu) - 1)/(1-mu))
                elif c_t < 0:
                    u = -1/(1-mu)
                esperanca = 0
                for w in range(N):
                    esperanca = esperanca + probabilidades[w]*V_old[t,w]
                valor[:,t] = (u + beta*esperanca)
```

```
                if valor[:,t]< valor[:,t-1]:
                    break
            if valor[:,t]< valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t-1])
                indice_a = (t-1)
            elif valor[:,t] >= valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t])
                indice_a = (t)
            start = copy.deepcopy(indice_a)
            g[j,i] = copy.deepcopy(grid_a[indice_a])

    dif = np.abs(V_new - V_old)
    erro = np.max(np.max(dif))
    print(erro)
fig, ax = plt.subplots()
for i in range(N):
    ax.plot(grid_a, V_new[:,i], color=plt.cm.jet(i / N), lw=2, alpha=0.6, label=f'Fu
ax.legend()
ax.set(xlim=(np.min(grid_a), np.max(grid_a)))
plt.show()
fig, ax = plt.subplots()
for i in range(N):
    ax.plot(grid_a, g[:,i], color=plt.cm.tab20c(i / N), lw=2, alpha=0.6, label=f'Fun
ax.legend()
ax.set(xlim=(np.min(grid_a), np.max(grid_a)))
plt.show()
fig, ax = plt.subplots()
for i in range(N):
    ax.plot(grid_a, -g[:,i]+(1+r)*grid_a+np.exp(z_grid[i]), color=plt.cm.tab20b(i /
ax.legend()
ax.set(xlim=(np.min(grid_a), np.max(grid_a)))
plt.show()
```

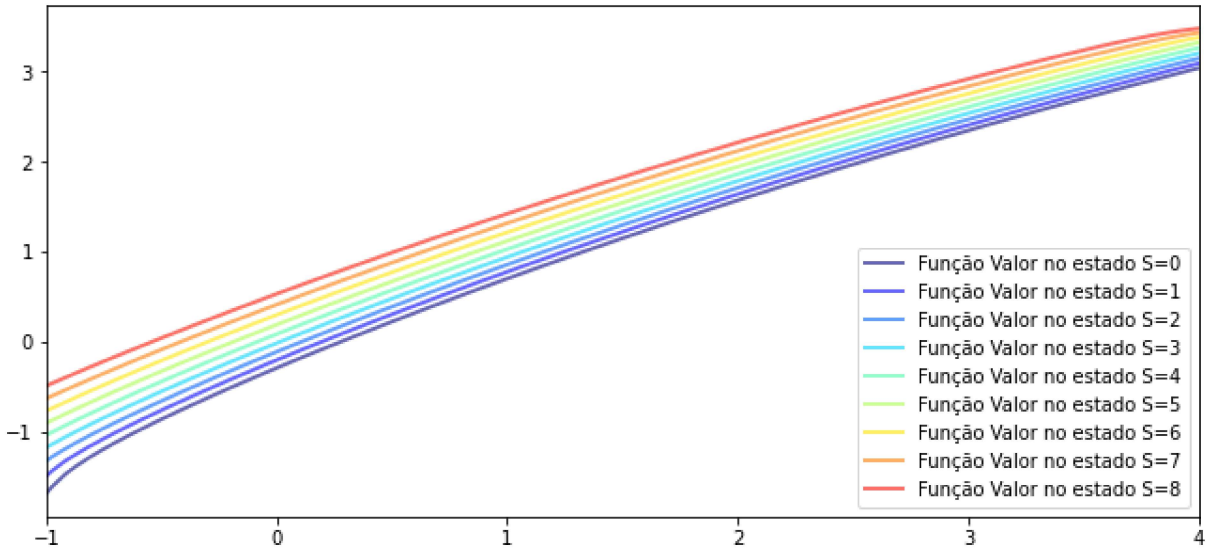
```
[-1.    -0.975 -0.95  -0.925 -0.9   -0.875 -0.85  -0.825 -0.8   -0.775
 -0.75  -0.725 -0.7   -0.675 -0.65  -0.625 -0.6   -0.575 -0.55  -0.525
 -0.5   -0.475 -0.45  -0.425 -0.4   -0.375 -0.35  -0.325 -0.3   -0.275
 -0.25  -0.225 -0.2   -0.175 -0.15  -0.125 -0.1   -0.075 -0.05  -0.025
  0.     0.025  0.05   0.075  0.1    0.125  0.15   0.175  0.2    0.225
  0.25   0.275  0.3    0.325  0.35   0.375  0.4    0.425  0.45   0.475
  0.5    0.525  0.55   0.575  0.6    0.625  0.65   0.675  0.7    0.725
  0.75   0.775  0.8    0.825  0.85   0.875  0.9    0.925  0.95   0.975
  1.     1.025  1.05   1.075  1.1    1.125  1.15   1.175  1.2    1.225
  1.25   1.275  1.3    1.325  1.35   1.375  1.4    1.425  1.45   1.475
  1.5    1.525  1.55   1.575  1.6    1.625  1.65   1.675  1.7    1.725
  1.75   1.775  1.8    1.825  1.85   1.875  1.9    1.925  1.95   1.975
  2.     2.025  2.05   2.075  2.1    2.125  2.15   2.175  2.2    2.225
  2.25   2.275  2.3    2.325  2.35   2.375  2.4    2.425  2.45   2.475
  2.5    2.525  2.55   2.575  2.6    2.625  2.65   2.675  2.7    2.725
  2.75   2.775  2.8    2.825  2.85   2.875  2.9    2.925  2.95   2.975
  3.     3.025  3.05   3.075  3.1    3.125  3.15   3.175  3.2    3.225
  3.25   3.275  3.3    3.325  3.35   3.375  3.4    3.425  3.45   3.475
  3.5    3.525  3.55   3.575  3.6    3.625  3.65   3.675  3.7    3.725
  3.75   3.775  3.8    3.825  3.85   3.875  3.9    3.925  3.95   3.975
  4.   ]
0.5180268562582369
0.42518755100743716
0.35878265006491916
0.3004793667388097
0.2521028737762654
0.21264477299240392
0.18050947510148374
0.15424131575757105
0.13260982307184577
0.11471134019888618
```

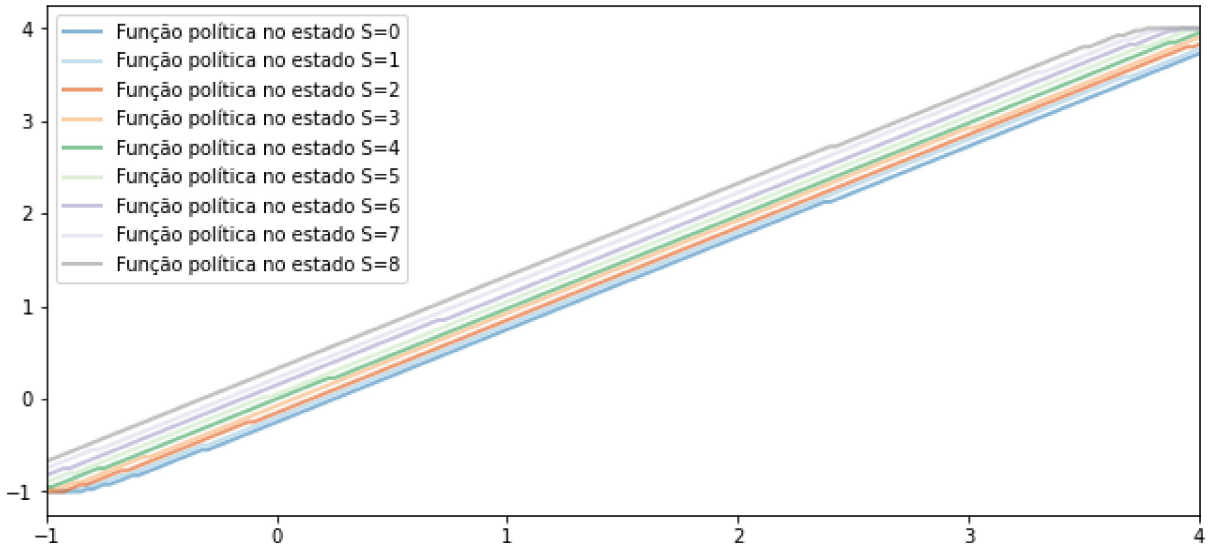
```
0.09970241980523076
0.08706233747664704
0.07645996127429733
0.0670522327599361
0.058802633304807284
0.05172791833855772
0.045674495390779946
0.04046591182639592
0.035896950576258746
0.03193161301771763
0.028489719702883498
0.025448354349642877
0.022765377857134617
0.020454235486778494
0.01833566918116425
0.016472256394512108
0.014840333051740373
0.013358075207032805
0.012145485501385167
0.011615977881410222
0.011111633005694843
0.010630653863937134
0.010172024397829382
0.009734471462357508
0.00931682805081202
0.008918154268453549
0.00853750350596871 3
0.00817394127928317
0.007826641210338714
0.0074948157766323575
0.007177700519972685
0.006874567124824926
0.006584728890987046
0.006307544125082387
0.006042276024054827
0.00578849487682076
0.005545650227058108
0.005313302566031464
0.005091014878799038
0.004878314755881075
0.004674735688236176
0.004479846760780504
0.004293259584241982
0.004114624468190886
0.003943590805521202
0.0037798209827437645
0.003622993280412601
0.0034728006805606615
0.00332895177452785 5
0.0031911697836481956
0.0030591882859707376
0.002932754302743090 2
0.00281163058507427 6
0.0026955901949137218
0.00258441431095501 9
0.0024778931271067783
0.0023758264530238105
0.0022780232840848846
0.0021843011064341944
0.0020944854804902935
0.0020084098387709304
0.0019259151093542481
0.0018468492620677956
0.001771066897638729
```

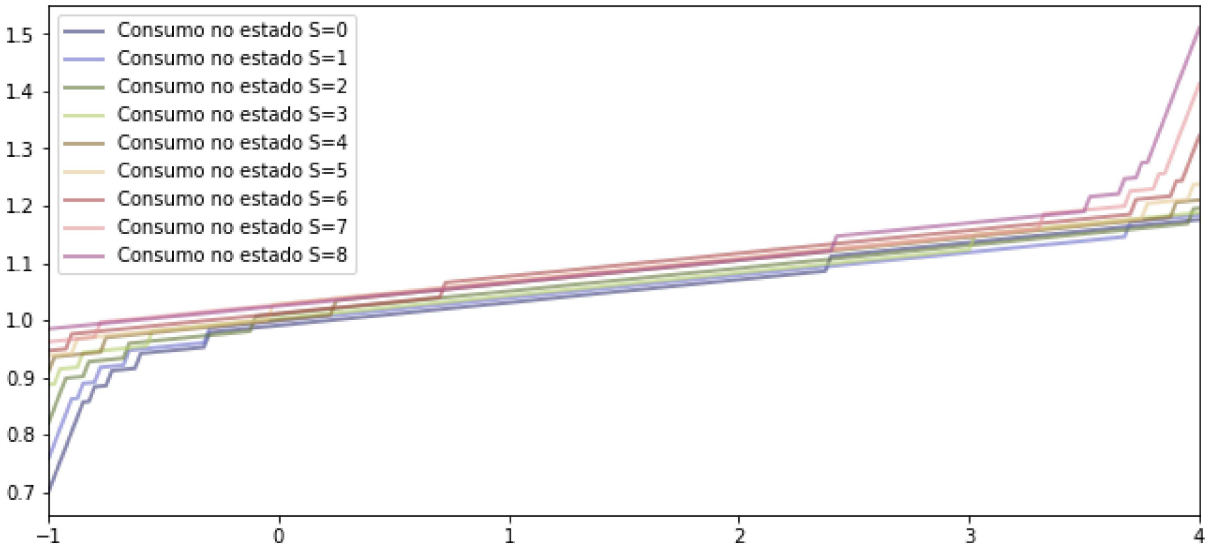
```
0.0016984290465595908
0.0016288028358268924
0.0015620612068270034
0.0014980826461106211
0.001436750907282569
0.0013779547649752644
0.0013215877634056294
0.001267547997741314
0.0012157379458788142
0.0011660642997102233
0.0011184377746582541
0.0010727729188992896
0.0010289879283389425
0.00098700446747646
0.0009467475066891051
0.0009081451638484062
0.0008711285869642538
0.00083563136615461
0.0008015903768505694
0.0007689455746882068
0.0007376393317111152
0.0007076162948356224
0.0006788233607020189
0.0006512096460640215
0.000624724726401595721
0.0005993269208310892
0.0005749664548821976
0.0005516021295073337
0.0005291928636692589
0.0005076992928847446
0.0004870836953136948
0.0004673096330178872
0.0004483418068121914
0.0004301480355617393
0.0004126966831548007
0.0003959571614124968
0.0003799001637778687
0.00036449761751455867
0.0003497226313153856
0.0003355494393306735
0.0003219533523113327
0.0003089107087310694
0.00029639883012078094
0.0002843959578322749
0.0002728813151187115
0.00026183486382347354
0.00025123747283917197
0.0002410707807808033763
0.0002313171833883132
0.00022195980164796403
0.00021298245171319508
0.000204369615786959
0.00019610641440781507
0.00018817857987052378
0.0001805724307828438
0.0001732748477540902
0.0001662732501177544
0.00015955557359870198
0.00015311024894626435
0.00014692618144751535
0.00014099273129741796
0.00013529969479098014
0.00012983728628546132
0.00012459612090776062
```

```
0.00011956719798611459
0.00011474188520432804
0.00011011190341347898
0.00010566931205624464
0.00010140649523249223
9.731614832597657e-05
9.339126519414265e-05
8.962512588039928e-05
8.60112848410921e-05
8.25435596614188e-05
7.921602023053254e-05
7.60229783702826e-05
7.295897789916239e-05
7.001878509660386e-05
6.719737957228133e-05
6.448994550511955e-05
6.189186325800122e-05
5.939870132509739e-05
5.700620862292283e-05
5.4710307097138866e-05
5.2507084630670775e-05
5.039278825735849e-05
4.836381763473341e-05
4.641671879990206e-05
4.454817818944079e-05
4.2755016884665764e-05
4.1034185122246214e-05
3.938275701309557e-05
3.7797925487748074e-05
3.627699745312185e-05
3.4817389132690835e-05
3.3416621617154973e-05
3.207231658297616e-05
3.078219218854983e-05
2.9544059145347745e-05
2.835581694338174e-05
2.721545023187666e-05
2.6121025353376126e-05
2.507068701218529e-05
2.4062655088030738e-05
2.309522157006 8603e-05
2.216674762900439e-05
2.127566079579 4238e-05
2.0420452267577716e-05
1.9599674315973203e-05
1.8811937805063295e-05
1.805590980907823e-05
1.7330311330443493e-05
1.663391510775547e-05
1.5965543516127667e-05
1.5324066546806847e-05
1.4708399882934486e-05
1.4117503037924806e-05
1.3550377581328377e-05
1.3006065429088665e-05
1.2483647211514182e-05
1.1982240700758595e-05
1.1500999305358306e-05
1.1039110627164561e-05
1.0595795076007164e-05
1.0170304536316621e-05
9.761921098139226e-06
9.369955831228793e-06
8.99374760887639e-06
```

```
8.632661987029167e-06
8.286090122044953e-06
7.953447734632135e-06
7.634174114201642e-06
7.327731169182172e-06
7.033602511086201e-06
6.751292572326761e-06
6.48032576933133e-06
6.220245694743554e-06
5.970614338490776e-06
5.73101134726528e-06
5.5010333077643025e-06
5.2802930672335435e-06
5.068419072662422e-06
4.865054740399444e-06
4.669857855965631e-06
4.482499986524502e-06
4.3026659319878036e-06
4.130053184558946e-06
3.964371422693347e-06
3.8053420121642034e-06
3.652697539324734e-06
3.506181359025362e-06
3.3655471531890413e-06
3.230558523803495e-06
3.100988584581188e-06
2.976619577044204e-06
2.8572425052608708e-06
2.7426567825727943e-06
2.632669892088657e-06
2.5270970644974966e-06
2.425760964097634e-06
2.3284913921450823e-06
2.235124998639648e-06
2.14550500299282e-06
2.0594809353458032e-06
1.9769083747789296e-06
1.8976487117239316e-06
1.8215689090439469e-06
1.7485412802109579e-06
1.6784432714800346e-06
1.6111572556098963e-06
1.546570335131392e-06
1.4845741502789167e-06
1.4250646942493006e-06
1.3679421440038197e-06
1.3131106846309137e-06
1.2604783525826946e-06
1.2099568791335003e-06
1.1614615433863662e-06
1.1149110266117646e-06
1.070227277466529e-06
1.0273353798773144e-06
9.86163428029485e-07
```







Q1(c)

In [94]:

```
pi_0 = np.full((a_size, N),(1/(N*a_size)))

tol_distrb = 10**(-6)
pi_old = copy.deepcopy(pi_0)
pi_new = copy.deepcopy(pi_0)
pi_dif = np.ones((a_size,N),dtype=float)
```

```
erro = 1
t=0
pi_iter = [pi_0]
while(erro>tol_distrb):
    t += 1
    for i in range(N):
        for j in range(a_size):
            aux = pi_old*(g == grid_a[j])
            pi_new[j,i] = sum(aux)@b[:,i]
            pi_dif[j,i] = (pi_new[j,i]-pi_old[j,i])
            pi_dif[j,i] = np.abs(pi_dif[j,i])
    erro = np.max(pi_dif)
    pi_old = copy.deepcopy(pi_new)

print(f'Convergência após t={t}')
```

```
Convergência após t=412
```

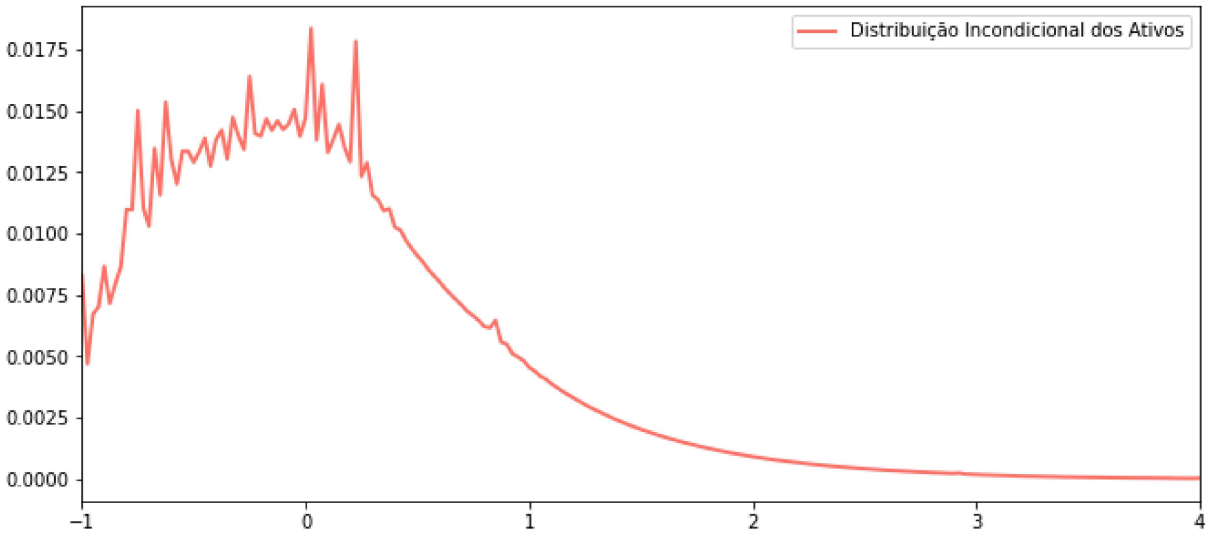
In [95]:

```
def asset_marginal(pi,a_size,N):
    asset_dist = np.zeros(a_size)
    for i in range(a_size):
        for j in range(N):
            asset_dist[i] += pi[i,j]
    return(asset_dist)
```

In [96]:

```
dist_a = asset_marginal(pi_new,a_size,N)

fig, ax = plt.subplots()
ax.plot(grid_a, dist_a, color=plt.cm.jet(i / N), lw=2, alpha=0.6, label=f'Distribuiç
ax.legend()
ax.set(xlim=(np.min(grid_a), np.max(grid_a)))
plt.show()
```



In [97]:

```
cdf_a = np.zeros(a_size)
for i in range(a_size):
    for j in range(i):
        cdf_a[i] += dist_a[j]

print(cdf_a)

fig, ax = plt.subplots()
ax.plot(grid_a, cdf_a, color=plt.cm.jet(i / N), lw=2, alpha=0.6, label=f'Distribuiçã
ax.legend()
```

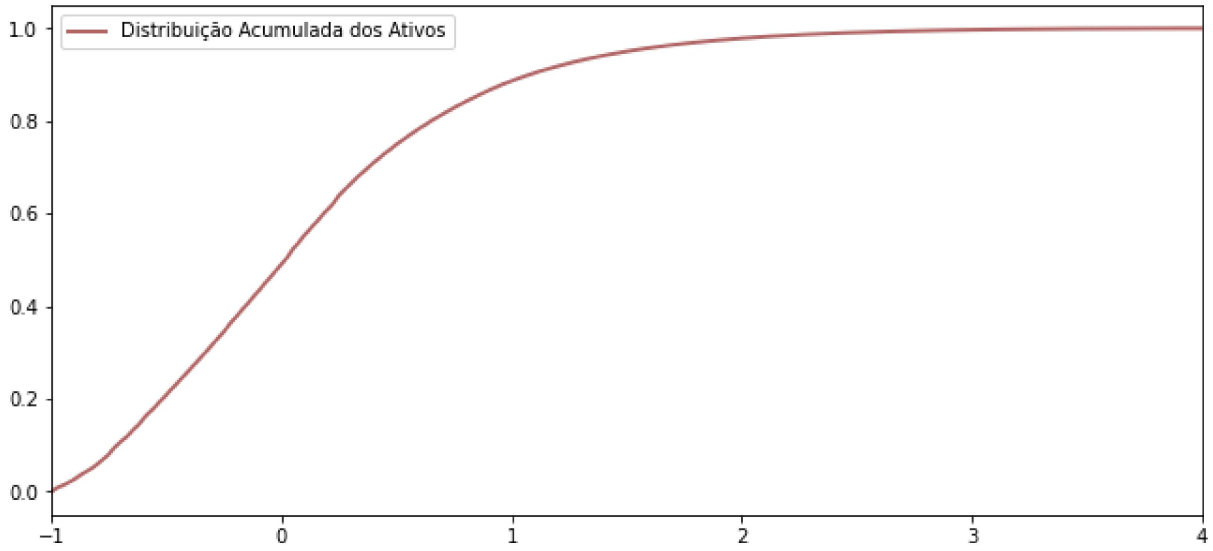
```
ax.set(xlim=(np.min(grid_a), np.max(grid_a)))
plt.show()

a_inf_0 = cdf_a[np.where(grid_a == 0)]
a_inf_1 = cdf_a[np.where(grid_a == 1)]
a_inf_2 = cdf_a[np.where(grid_a == 2)]
a_inf_3 = cdf_a[np.where(grid_a == 3)]

a_inf_0 = "{:.2%}".format(a_inf_0[0])
a_inf_1 = "{:.2%}".format(a_inf_1[0])
a_sup_2 = "{:.2%}".format(1-a_inf_2[0])
a_sup_3 = "{:.2%}".format(1-a_inf_3[0])


print(f'{a_inf_0} das famílias tem ativos negativos')
print(f'{a_inf_1} das famílias tem ativos entre -1 e 1')
print(f'Apenas {a_sup_2} das famílias tem ativos acima de 2')
print(f'Apenas {a_sup_3} das famílias tem ativos acima de 3')
```

```
[0.         0.00828588 0.01297424 0.01968302 0.02669642 0.0353504
 0.04250101 0.05044726 0.05911495 0.07009476 0.0810659  0.09607398
 0.10711152 0.117413   0.13088067 0.1424515  0.15780271 0.17084835
 0.18286919 0.19622184 0.20958146 0.22248248 0.23581501 0.2497008
 0.26244399 0.27625725 0.29046966 0.30350436 0.3182369  0.33222595
 0.34564855 0.36204287 0.37611225 0.39007476 0.40475052 0.41895041
 0.43354413 0.44778196 0.46224587 0.47730633 0.49126634 0.50597482
 0.52433859 0.53814388 0.55421571 0.5675207  0.58136357 0.59581141
 0.60931295 0.62223131 0.64006355 0.65238672 0.66527111 0.67683697
 0.68820525 0.69912692 0.71013705 0.72039394 0.7305303  0.74024348
 0.74962258 0.75871759 0.767554   0.7760765  0.78434932 0.79237734
 0.80012683 0.80763827 0.81492885 0.82199225 0.82880563 0.83544985
 0.84189696 0.84810065 0.85425189 0.8607086  0.86628543 0.87177388
 0.87687655 0.88184669 0.88666323 0.89120634 0.89559424 0.89977768
 0.90384099 0.9077091  0.9114256  0.91498441 0.91840869 0.92169835
 0.92484845 0.92787355 0.93077469 0.93356622 0.9362422  0.93881195
 0.94127796 0.94364735 0.94592257 0.9481065  0.95020375 0.95221764
 0.9541525  0.95601028 0.95779464 0.95950835 0.96115476 0.96273626
 0.96425543 0.96571484 0.96711695 0.96846414 0.96975848 0.9710021
 0.97219712 0.97334542 0.97444893 0.97550945 0.97652863 0.97750807
 0.97844973 0.97935428 0.98022383 0.98105986 0.98186306 0.98263536
 0.98337905 0.98409225 0.98477783 0.98543677 0.98607023 0.98667916
 0.98726456 0.98782731 0.98836834 0.98888849 0.98938853 0.98986933
 0.99033159 0.99077594 0.99120332 0.99161412 0.99200901 0.99238899
 0.9927541  0.993105   0.99344306 0.99376723 0.99407914 0.99438027
 0.99467044 0.99494788 0.99521743 0.99547047 0.99571648 0.99595619
 0.99617967 0.99640124 0.99663578 0.99682828 0.99701398 0.99718557
 0.99735034 0.99750669 0.99765421 0.99779461 0.99792742 0.99805406
 0.99817404 0.99828801 0.99839612 0.99849874 0.99859638 0.99868883
 0.99877684 0.99886017 0.99893951 0.99901459 0.99908597 0.9991536
 0.9992185  0.99927962 0.99933974 0.99939429 0.99944629 0.99949493
 0.99954082 0.99958498 0.99962612 0.99966631 0.99970272 0.99973736
 0.99977033 0.99981001 0.99984367 0.99987391 0.99989586 0.99991729
 0.99993373 0.99995126 0.99996953]
```




```
49.13% das famílias tem ativos negativos
88.67% das famílias tem ativos entre -1 e 1
Apenas 2.16% das famílias tem ativos acima de 2
Apenas 0.30% das famílias tem ativos acima de 3
```

Q1 (d)

In [98]:

```
# CALCULAR DEMANDA POR CAPITAL

demanda = 0

for i in range(N):
    for j in range(a_size):
        demanda += g[j,i]*pi_new[j,i]


print(demanda)
```

```
0.10567905266217047
```

Q1 (e)

i.

In [99]:

```
N = 9
m = 3
mu = 5
beta = 0.96
phi = 1
rho = 0.3
sigma_sq_er = 0.01 * (1-(rho)**2)
sigma_sq_z = sigma_sq_er/(1-rho**2)
zn = m*(sigma_sq_z)**(1/2)
z0 = -zn
s = (zn-z0)/(N-1)
z_grid = np.arange(z0,zn+s,s)
b = np.zeros((9,9))
for j in range(N):
    for i in range(N):
        if j==0:
            b[i,j] = norm.cdf((z0-rho*z_grid[i]+s/2)/sigma_sq_er**(1/2))
        elif j==N-1:
```

```
            b[i,j] = 1-norm.cdf((zn-rho*z_grid[i]-s/2)/sigma_sq_er**(1/2))
        else:
            b[i,j] = norm.cdf((z_grid[j] - rho*z_grid[i] + s/2)/sigma_sq_er**(1/2))

a_size=201
valor = np.zeros((1,a_size))
# ORDEM DA FUNÇÃO VALOR: LINHA É a, COLUNA É Z
a_size=201
a_max=4
tol = 10**(-6)
r=0.04
V_0 = np.zeros((a_size, N))
grid_a = np.linspace(-phi, a_max, a_size)
V_new = copy.deepcopy(V_0)
erro = 1
g = np.zeros((a_size, N))
t= 1
while(erro > tol):
    V_old = copy.deepcopy(V_new)
    t += 1
    for i in range(N):
        probabilidades = b[i,:]
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = 0
            valor = np.zeros((1,a_size))
            t = copy.deepcopy(start)
            c_t = (estoque - grid_a[t])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca = 0
            for w in range(N):
                esperanca = (esperanca + probabilidades[w]*V_old[t,w])
            valor[:,t] = (u + beta*esperanca)
            for t in range((start+1), a_size):
                c_t = (estoque - grid_a[t])
                if c_t >= 0:
                    u = ((c_t**(1-mu) - 1)/(1-mu))
                elif c_t < 0:
                    u = -1/(1-mu)
                esperanca = 0
                for w in range(N):
                    esperanca = esperanca + probabilidades[w]*V_old[t,w]
                valor[:,t] = (u + beta*esperanca)
                if valor[:,t]< valor[:,t-1]:
                    break
            if valor[:,t]< valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t-1])
                indice_a = (t-1)
            elif valor[:,t] >= valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t])
                indice_a = (t)
            start = copy.deepcopy(indice_a)
            g[j,i] = copy.deepcopy(grid_a[indice_a])

    dif = np.abs(V_new - V_old)
    erro = np.max(np.max(dif))
pi_0 = np.full((a_size, N),(1/(N*a_size)))

tol_distrb = 10**(-6)
pi_old = copy.deepcopy(pi_0)
```

```
pi_new = copy.deepcopy(pi_0)
pi_dif = np.ones((a_size,N),dtype=float)
erro = 1
t=0
pi_iter = [pi_0]
while(erro>tol_distrb):
    t += 1
    for i in range(N):
        for j in range(a_size):
            aux = pi_old*(g == grid_a[j])
            pi_new[j,i] = sum(aux)@b[:,i]
            pi_dif[j,i] = (pi_new[j,i]-pi_old[j,i])
            pi_dif[j,i] = np.abs(pi_dif[j,i])
    erro = np.max(pi_dif)
    pi_old = copy.deepcopy(pi_new)
# CALCULAR DEMANDA POR CAPITAL

demanda2 = 0

for i in range(N):
    for j in range(a_size):
        demanda2 += g[j,i]*pi_new[j,i]

if demanda2>demanda:
    sinal = 'um aumento'
else:
    sinal = 'uma diminuição'

print(f'Nosso excesso de demanda é de {demanda2}, {sinal} de {abs(demanda2-demanda)}
```

```
Nosso excesso de demanda é de 1.193686529578574, um aumento de 1.0880074769164034. O
riginalmente tínhamos 0.10567905266217047.
```

ii.

In [ ]:

```
N = 9
m = 3
mu = 3
beta = 0.96
phi = 1
rho = 0.6
sigma_sq_er = 0.01 * (1-(rho)**2)
sigma_sq_z = sigma_sq_er/(1-rho**2)
zn = m*(sigma_sq_z)**(1/2)
z0 = -zn
s = (zn-z0)/(N-1)
z_grid = np.arange(z0,zn+s,s)
b = np.zeros((9,9))
for j in range(N):
    for i in range(N):
        if j==0:
            b[i,j] = norm.cdf((z0-rho*z_grid[i]+s/2)/sigma_sq_er**(1/2))
        elif j==N-1:
            b[i,j] = 1-norm.cdf((zn-rho*z_grid[i]-s/2)/sigma_sq_er**(1/2))
        else:
            b[i,j] = norm.cdf((z_grid[j] - rho*z_grid[i] + s/2)/sigma_sq_er**(1/2))

a_size=201
valor = np.zeros((1,a_size))
# ORDEM DA FUNÇÃO VALOR: LINHA É a, COLUNA É Z
a_size=201
a_max=4
tol = 10**(-6)
```

```
r=0.04
V_0 = np.zeros((a_size, N))
grid_a = np.linspace(-phi, a_max, a_size)
V_new = copy.deepcopy(V_0)
erro = 1
g = np.zeros((a_size, N))
t= 1
while(erro > tol):
    V_old = copy.deepcopy(V_new)
    t += 1
    for i in range(N):
        probabilidades = b[i,:]
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = 0
            valor = np.zeros((1,a_size))
            t = copy.deepcopy(start)
            c_t = (estoque - grid_a[t])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca = 0
            for w in range(N):
                esperanca = (esperanca + probabilidades[w]*V_old[t,w])
            valor[:,t] = (u + beta*esperanca)
            for t in range((start+1), a_size):
                c_t = (estoque - grid_a[t])
                if c_t >= 0:
                    u = ((c_t**(1-mu) - 1)/(1-mu))
                elif c_t < 0:
                    u = -1/(1-mu)
                esperanca = 0
                for w in range(N):
                    esperanca = esperanca + probabilidades[w]*V_old[t,w]
                valor[:,t] = (u + beta*esperanca)
                if valor[:,t]< valor[:,t-1]:
                    break
            if valor[:,t]< valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t-1])
                indice_a = (t-1)
            elif valor[:,t] >= valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t])
                indice_a = (t)
            start = copy.deepcopy(indice_a)
            g[j,i] = copy.deepcopy(grid_a[indice_a])

    dif = np.abs(V_new - V_old)
    erro = np.max(np.max(dif))
pi_0 = np.full((a_size, N),(1/(N*a_size)))

tol_distrb = 10**(-6)
pi_old = copy.deepcopy(pi_0)
pi_new = copy.deepcopy(pi_0)
pi_dif = np.ones((a_size,N),dtype=float)
erro = 1
t=0
pi_iter = [pi_0]
while(erro>tol_distrb):
    t += 1
    for i in range(N):
        for j in range(a_size):
            aux = pi_old*(g == grid_a[j])
```

```
            pi_new[j,i] = sum(aux)@b[:,i]
            pi_dif[j,i] = (pi_new[j,i]-pi_old[j,i])
            pi_dif[j,i] = np.abs(pi_dif[j,i])
    erro = np.max(pi_dif)
    pi_old = copy.deepcopy(pi_new)
# CALCULAR DEMANDA POR CAPITAL

demanda3 = 0

for i in range(N):
    for j in range(a_size):
        demanda3 += g[j,i]*pi_new[j,i]

if demanda3>demanda:
    sinal = 'um aumento'
else:
    sinal = 'uma diminuição'

print(f'Nosso excesso de demanda é de {demanda3}, {sinal} de {abs(demanda3-demanda)}
```

iii.

In [ ]:

```
N = 9
m = 3
mu = 3
beta = 0.96
phi = 1
rho = 0.3
sigma_sq_er = 0.04 * (1-(rho)**2)
sigma_sq_z = sigma_sq_er/(1-rho**2)
zn = m*(sigma_sq_z)**(1/2)
z0 = -zn
s = (zn-z0)/(N-1)
z_grid = np.arange(z0,zn+s,s)
b = np.zeros((9,9))
for j in range(N):
    for i in range(N):
        if j==0:
            b[i,j] = norm.cdf((z0-rho*z_grid[i]+s/2)/sigma_sq_er**(1/2))
        elif j==N-1:
            b[i,j] = 1-norm.cdf((zn-rho*z_grid[i]-s/2)/sigma_sq_er**(1/2))
        else:
            b[i,j] = norm.cdf((z_grid[j] - rho*z_grid[i] + s/2)/sigma_sq_er**(1/2))

a_size=201
valor = np.zeros((1,a_size))
# ORDEM DA FUNÇÃO VALOR: LINHA É a, COLUNA É Z
a_size=201
a_max=4
tol = 10**(-6)
r=0.04
V_0 = np.zeros((a_size, N))
grid_a = np.linspace(-phi, a_max, a_size)
V_new = copy.deepcopy(V_0)
erro = 1
g = np.zeros((a_size, N))
t= 1
while(erro > tol):
    V_old = copy.deepcopy(V_new)
    t += 1
    for i in range(N):
        probabilidades = b[i,:]
```

```
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = 0
            valor = np.zeros((1,a_size))
            t = copy.deepcopy(start)
            c_t = (estoque - grid_a[t])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca = 0
            for w in range(N):
                esperanca = (esperanca + probabilidades[w]*V_old[t,w])
            valor[:,t] = (u + beta*esperanca)
            for t in range((start+1), a_size):
                c_t = (estoque - grid_a[t])
                if c_t >= 0:
                    u = ((c_t**(1-mu) - 1)/(1-mu))
                elif c_t < 0:
                    u = -1/(1-mu)
                esperanca = 0
                for w in range(N):
                    esperanca = esperanca + probabilidades[w]*V_old[t,w]
                valor[:,t] = (u + beta*esperanca)
                if valor[:,t]< valor[:,t-1]:
                    break
            if valor[:,t]< valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t-1])
                indice_a = (t-1)
            elif valor[:,t] >= valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t])
                indice_a = (t)
            start = copy.deepcopy(indice_a)
            g[j,i] = copy.deepcopy(grid_a[indice_a])

    dif = np.abs(V_new - V_old)
    erro = np.max(np.max(dif))
pi_0 = np.full((a_size, N),(1/(N*a_size)))

tol_distrb = 10**(-6)
pi_old = copy.deepcopy(pi_0)
pi_new = copy.deepcopy(pi_0)
pi_dif = np.ones((a_size,N),dtype=float)
erro = 1
t=0
pi_iter = [pi_0]
while(erro>tol_distrb):
    t += 1
    for i in range(N):
        for j in range(a_size):
            aux = pi_old*(g == grid_a[j])
            pi_new[j,i] = sum(aux)@b[:,i]
            pi_dif[j,i] = (pi_new[j,i]-pi_old[j,i])
            pi_dif[j,i] = np.abs(pi_dif[j,i])
    erro = np.max(pi_dif)
    pi_old = copy.deepcopy(pi_new)
# CALCULAR DEMANDA POR CAPITAL

demanda4 = 0

for i in range(N):
    for j in range(a_size):
        demanda4 += g[j,i]*pi_new[j,i]
```

```
if demanda4>demanda:
    sinal = 'um aumento'
else:
    sinal = 'uma diminuição'

print(f'Nosso excesso de demanda é de {demanda4}, {sinal} de {abs(demanda4-demanda)}
```

Q1 (bonus)

In [ ]:

```
N = 9
m = 3
mu = 3
beta = 0.96
phi = 1
rho = 0.3
sigma_sq_er = 0.01 * (1-(rho)**2)
sigma_sq_z = sigma_sq_er/(1-rho**2)
```

In [ ]:

```
# Repetindo o código da questão 1 (pois ele não vai mudar a cada loop)
zn = m*(sigma_sq_z)**(1/2)
z0 = -zn
s = (zn-z0)/(N-1)
z_grid = np.arange(z0,zn+s,s)
b = np.zeros((9,9))
for j in range(N):
    for i in range(N):
        if j==0:
            b[i,j] = norm.cdf((z0-rho*z_grid[i]+s/2)/sigma_sq_er**(1/2))
        elif j==N-1:
            b[i,j] = 1-norm.cdf((zn-rho*z_grid[i]-s/2)/sigma_sq_er**(1/2))
        else:
            b[i,j] = norm.cdf((z_grid[j] - rho*z_grid[i] + s/2)/sigma_sq_er**(1/2))

a_size=201
a_max=4
tol = 10**(-6)
grid_a = np.linspace(-phi, a_max, a_size)
```

In [ ]:

```
R_inf = 0
R_sup = 1.04 #1+r = R

demanda_inf = -1
demanda_sup = copy.deepcopy(demanda)

# Vamos agora fazer o loop para o método da bissecção
cont = 0
demanda_abs=1
while (demanda_abs>10**(-3)):
    r = (-demanda_inf*R_sup + demanda_sup*R_inf)/(demanda_sup-demanda_inf) - 1

    V_0 = np.zeros((a_size, N))
    V_new = copy.deepcopy(V_0)
    erro = 1
    g = np.zeros((a_size, N))
```

```
t= 1
while(erro > tol):
    V_old = copy.deepcopy(V_new)
    t += 1
    for i in range(N):
        probabilidades = b[i,:]
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = 0
            valor = np.zeros((1,a_size))
            t = copy.deepcopy(start)
            c_t = (estoque - grid_a[t])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca = 0
            for w in range(N):
                esperanca = (esperanca + probabilidades[w]*V_old[t,w])
            valor[:,t] = (u + beta*esperanca)
            for t in range((start+1), a_size):
                c_t = (estoque - grid_a[t])
                if c_t >= 0:
                    u = ((c_t**(1-mu) - 1)/(1-mu))
                elif c_t < 0:
                    u = -1/(1-mu)
                esperanca = 0
                for w in range(N):
                    esperanca = esperanca + probabilidades[w]*V_old[t,w]
                valor[:,t] = (u + beta*esperanca)
                if valor[:,t]< valor[:,t-1]:
                    break
            if valor[:,t]< valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t-1])
                indice_a = (t-1)
            elif valor[:,t] >= valor[:,t-1]:
                V_new[j,i] = copy.deepcopy(valor[:,t])
                indice_a = (t)
            start = copy.deepcopy(indice_a)
            g[j,i] = copy.deepcopy(grid_a[indice_a])

    dif = np.abs(V_new - V_old)
    erro = np.max(np.max(dif))
pi_0 = np.full((a_size, N),(1/(N*a_size)))

tol_distrb = 10**(-6)
pi_old = copy.deepcopy(pi_0)
pi_new = copy.deepcopy(pi_0)
pi_dif = np.ones((a_size,N),dtype=float)
erro = 1
t=0
pi_iter = [pi_0]
while(erro>tol_distrb):
    t += 1
    for i in range(N):
        for j in range(a_size):
            aux = pi_old*(g == grid_a[j])
            pi_new[j,i] = sum(aux)@b[:,i]
            pi_dif[j,i] = (pi_new[j,i]-pi_old[j,i])
            pi_dif[j,i] = np.abs(pi_dif[j,i])
    erro = np.max(pi_dif)
    pi_old = copy.deepcopy(pi_new)
# CALCULAR DEMANDA POR CAPITAL
```

```
    demanda_new = 0
    cont += 1

    for i in range(N):
        for j in range(a_size):
            demanda_new += g[j,i]*pi_new[j,i]
    demanda_abs=abs(copy.deepcopy(demanda_new))
    if demanda_new>0:
        R_sup = copy.deepcopy(1+r)
        demanda_sup = copy.deepcopy(demanda_new)
        print(f'{cont})r = {r}, o excesso de de demanda é de {demanda_sup}')
    elif demanda_new<0:
        R_inf = copy.deepcopy(1+r)
        demanda_inf = copy.deepcopy(demanda_new)
        print(f'{cont})r = {r}, o excesso de de demanda é de {demanda_inf}')

print(f'A taxa de juros que equilibra o mercado é {r}')
print(demanda_new)
```

Q2

In [100…

```
N = 9
m = 3
mu = 3
beta = 0.96
phi = 1
rho = 0.3
sigma_sq_er = 0.01 * (1-(rho)**2)
sigma_sq_z = sigma_sq_er/(1-rho**2)
zn = m*(sigma_sq_z)**(1/2)
z0 = -zn
s = (zn-z0)/(N-1)
z_grid = np.arange(z0,zn+s,s)
print(z_grid)
b = np.zeros((9,9))
for j in range(N):
    for i in range(N):
        if j==0:
            b[i,j] = norm.cdf((z0-rho*z_grid[i]+s/2)/sigma_sq_er**(1/2))
        elif j==N-1:
            b[i,j] = 1-norm.cdf((zn-rho*z_grid[i]-s/2)/sigma_sq_er**(1/2))
        else:
            b[i,j] = norm.cdf((z_grid[j] - rho*z_grid[i] + s/2)/sigma_sq_er**(1/2))
```

```
[-0.3   -0.225 -0.15  -0.075  0.     0.075  0.15   0.225  0.3  ]
```

In [101…

```
# Fazendo a calibração da demografia
data = pd.read_excel('LifeTable.xlsx')

# Vamos eliminar as variáveis Death probability e Life expatancy
data.drop(['Death probability - male', 'Life expectancy - male','Death  probability

# Retirando os individuos com mais de 87 anos e menos de 18
data.drop(range(88,120),axis = 0, inplace = True)
data.drop(range(18),axis = 0, inplace = True)


# Combinando o numero de homens e mulheres vivos
```

```
sum_col = data["Number of lives - male"]+data["Number of lives - male"]
data["Number"] = sum_col


data.drop(["Number of lives - male","Number of lives - female"], axis = 1, inplace =


# Vamos definir nossa medida
measure = data.iloc[0:45,1].sum()/45
print(measure)

# Vamos agora preencher o vetor surv, que no dá a probabilidade de continuar vivo ao
# por definição, para todo t menor ou igual a 8, st = 1
surv = np.ones(14)
s = np.ones(14)
for i in range(9,13):
    s[i] = data.iloc[(i*5+6):(5*i+10),1].sum()/(5*measure)
for j in range(9,13):
    surv[j] = s[j]/s[j-1]
surv[13] = 0

print(surv)
```

```
188.22853333333333
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         0.61257875 0.8681616  0.79595457
 0.68210388 0.        ]
```

In [102…

```
a_size=201
a_max=4
a_min=-1
phi1 = -9.5
phi2 = 11.6
r = 0.04
grid_a = np.linspace(-phi, a_max, a_size)

psi = np.zeros(a_size)
def psi(i):
    psi = phi1 *(1 + grid_a[i]/phi2)**(1-mu)
    return(psi)
```

In [103…

```
tol = 10**(-6)
previdencia =0.4
# Gerando a função valor e a função política dos aposentados
V_ret = np.zeros((a_size, 5))
a_ret = np.zeros((a_size, 5))

# Preenchendo a ultima coluna de V_ret

for i in range(a_size):
    estoque = copy.deepcopy((1+r)*grid_a[i]+previdencia)
    if i==0:
        start = copy.deepcopy(0)
    valor = np.zeros((1,a_size))
    c = (estoque - grid_a[start])
    if c>=0:
        u_c = ((c**(1-mu) - 1)/(1-mu))
    elif c < 0:
        u_c = -1/(1-mu)
    valor[:,(start)] = u_c + beta*psi(start)
    for j in range((start+1),a_size):
```

```
        c = (estoque - grid_a[j])
        if c >= 0:
            u_c = ((c**(1-mu) - 1)/(1-mu))
        elif c < 0:
            u_c = -1/(1-mu)
        valor[:,j] = u_c + beta*psi(j)
        if valor[:,j]<valor[:,(j-1)]:
            break
    if valor[:,j]<valor[:,(j-1)]:
        V_ret[i,4] = copy.deepcopy(valor[:,(j-1)])
        indice_a = copy.deepcopy(j-1)
    elif valor[:,j]>=valor[:,(j-1)]:
        V_ret[i,4] = copy.deepcopy(valor[:,j])
        indice_a = copy.deepcopy(j)
    start=copy.deepcopy(indice_a)
    a_ret[i,4] = copy.deepcopy(grid_a[indice_a])
print(a_ret[:,4])
```

```
[-1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.    -1.
 -1.    -1.    -1.    -1.    -1.    -1.    -1.    -0.975 -0.95  -0.925
 -0.9   -0.9   -0.875 -0.85  -0.825 -0.8   -0.775 -0.75  -0.725 -0.7
 -0.675 -0.65  -0.625 -0.6   -0.575 -0.55  -0.525 -0.5   -0.475 -0.45
 -0.425 -0.4   -0.375 -0.35  -0.325 -0.3   -0.275 -0.25  -0.225 -0.2
 -0.175 -0.15  -0.15  -0.125 -0.1   -0.075 -0.05  -0.025  0.     0.025
  0.05   0.075  0.1    0.125  0.15   0.175  0.2    0.225  0.25   0.275
  0.3    0.325  0.35   0.375  0.4    0.425  0.45   0.475  0.5    0.525
  0.55   0.575  0.6    0.625  0.625  0.65   0.675  0.7    0.725  0.75
  0.775  0.8    0.825  0.85   0.875  0.9    0.925  0.95   0.975  1.
  1.025  1.05   1.075  1.1    1.125  1.15   1.175  1.2    1.225  1.25
  1.275  1.3    1.325  1.35   1.375  1.375  1.4    1.425  1.45   1.475
  1.5    1.525  1.55   1.575  1.6    1.625  1.65   1.675  1.7    1.725
  1.75   1.775  1.8    1.825  1.85   1.875  1.9    1.925  1.95   1.975
  2.     2.025  2.05   2.075  2.1    2.125  2.125  2.15   2.175  2.2
  2.225  2.25   2.275  2.3    2.325  2.35   2.375  2.4    2.425  2.45
  2.475  2.5    2.525  2.55   2.575  2.6    2.625  2.65   2.675  2.7
  2.725  2.75   2.775  2.8    2.825  2.85   2.875  2.9    2.9    2.925
  2.95   2.975  3.     3.025  3.05   3.075  3.1    3.125  3.15   3.175
  3.2    3.225  3.25   3.275  3.3    3.325  3.35   3.375  3.4    3.425
  3.45 ]
```

In [104…

```
# Preenchendo os demais valores
for t in range(1,5):
    for i in range(a_size):
        estoque = copy.deepcopy((1+r)*grid_a[i]+previdencia)
        if i==0:
            start = copy.deepcopy(0)
        valor = np.zeros((1,a_size))
        c = (estoque - grid_a[start])
        if c>=0:
            u_c = ((c**(1-mu) - 1)/(1-mu))
        elif c < 0:
            u_c = -1/(1-mu)
        valor[:,(start)] = (u_c + beta*(surv[13-t]*V_ret[(start),(5-t)]+(1-surv[13-t
        for j in range((start+1),a_size):
            c = (estoque - grid_a[j])
            if c >= 0:
                u_c = ((c**(1-mu) - 1)/(1-mu))
            elif c < 0:
                u_c = -1/(1-mu)
            valor[:,j] = u_c + beta*(surv[13-t]*V_ret[j,(5-t)]+(1-surv[13-t])*psi(j)
            if valor[:,j]<valor[:,(j-1)]:
                break
        if valor[:,j]<valor[:,(j-1)]:
```

```
            V_ret[i,(4-t)] = copy.deepcopy(valor[:,(j-1)])
            indice_a = copy.deepcopy(j-1)
        elif valor[:,j]>=valor[:,(j-1)]:
            V_ret[i,(4-t)] = copy.deepcopy(valor[:,j])
            indice_a = copy.deepcopy(j)
        start=copy.deepcopy(indice_a)
        a_ret[i,(4-t)] = copy.deepcopy(grid_a[indice_a])

print(a_ret[:,0])

print(V_ret[:,2])
```

```
[-1.    -1.    -1.    -1.    -0.975 -0.95  -0.95  -0.925 -0.9   -0.875
 -0.875 -0.85  -0.825 -0.8   -0.8   -0.775 -0.75  -0.725 -0.7   -0.7
 -0.675 -0.65  -0.625 -0.6   -0.6   -0.575 -0.55  -0.525 -0.5   -0.5
 -0.475 -0.45  -0.425 -0.4   -0.4   -0.375 -0.35  -0.325 -0.3   -0.275
 -0.25  -0.25  -0.225 -0.2   -0.175 -0.15  -0.125 -0.125 -0.1   -0.075
 -0.05  -0.025  0.     0.025  0.025  0.05   0.075  0.1    0.125  0.15
  0.15   0.175  0.2    0.225  0.25   0.275  0.275  0.3    0.325  0.35
  0.375  0.4    0.425  0.45   0.475  0.5    0.5    0.525  0.55   0.575
  0.6    0.625  0.65   0.675  0.7    0.725  0.75   0.775  0.8    0.825
  0.85   0.875  0.875  0.9    0.925  0.95   0.975  1.     1.025  1.05
  1.075  1.1    1.125  1.15   1.175  1.2    1.225  1.25   1.275  1.3
  1.325  1.35   1.375  1.4    1.425  1.45   1.475  1.5    1.525  1.55
  1.575  1.6    1.625  1.65   1.675  1.7    1.725  1.75   1.75   1.775
  1.8    1.825  1.85   1.875  1.9    1.925  1.95   1.975  2.     2.025
  2.05   2.075  2.1    2.125  2.15   2.175  2.2    2.225  2.25   2.275
  2.3    2.325  2.35   2.375  2.4    2.425  2.45   2.475  2.5    2.525
  2.55   2.575  2.6    2.6    2.625  2.65   2.675  2.7    2.725  2.75
  2.775  2.8    2.825  2.85   2.875  2.9    2.925  2.95   2.975  3.
  3.025  3.05   3.075  3.1    3.125  3.15   3.175  3.2    3.225  3.25
  3.275  3.3    3.325  3.35   3.375  3.4    3.425  3.45   3.475  3.475
  3.5  ]
[-17.95064759 -17.44841768 -17.03686034 -16.62975077 -16.2917781
 -15.95720179 -15.66052231 -15.38118407 -15.10471376 -14.85818305
 -14.62433794 -14.39325845 -14.18543699 -13.98743593 -13.79232887
 -13.61487485 -13.4455036  -13.27927242 -13.12598662 -12.9797639
 -12.83697946 -12.7031769  -12.57587774 -12.45232932 -12.33441594
 -12.2227412  -12.11512256 -12.01030374 -11.91164683 -11.81728945
 -11.72340883 -11.63568822 -11.55054246 -11.4677932  -11.38933034
 -11.31197639 -11.23866585 -11.16762775 -11.09738114 -11.031814
 -10.9668248  -10.90331129 -10.84273398 -10.78462545 -10.72692518
 -10.67068107 -10.61779886 -10.56517548 -10.51340237 -10.46363638
 -10.41419877 -10.36508659 -10.31629694 -10.26775178 -10.21951908
 -10.17160152 -10.12399633 -10.07670073 -10.02971201  -9.98302745
  -9.93664439  -9.89056019  -9.8446984   -9.79909918  -9.75379254
  -9.70877592  -9.66404681  -9.61960272  -9.57544118  -9.53155976
  -9.48795605  -9.44453531  -9.40136113  -9.35845919  -9.31582716
  -9.27346274  -9.23136366  -9.18952765  -9.14795249  -9.10663598
  -9.06557593  -9.02477019  -8.98421661  -8.94391309  -8.90385753
  -8.86404787  -8.82445415  -8.78506931  -8.74592509  -8.70701951
  -8.6683506   -8.62991641  -8.591715    -8.55374445  -8.51600289
  -8.47848843  -8.44114111  -8.40399549  -8.36707252  -8.33037037
  -8.29388727  -8.25762143  -8.2215711   -8.18573454  -8.15011004
  -8.11466674  -8.07937025  -8.04428191  -8.00940005  -7.97472303
  -7.94024922  -7.90597699  -7.87190476  -7.83803094  -7.80435396
  -7.77087228  -7.73758435  -7.70448865  -7.67158369  -7.63886796
  -7.60629372  -7.57390529  -7.5417023   -7.50968332  -7.47784691
  -7.44619166  -7.41471619  -7.38341909  -7.352299    -7.32135456
  -7.29053786  -7.2598789   -7.22939235  -7.19907689  -7.16893123
  -7.13895407  -7.10914413  -7.07950016  -7.05002089  -7.02070509
  -6.99149003  -6.9624274   -6.93352542  -6.90478287  -6.87619856
  -6.84777132  -6.81949998  -6.79138336  -6.76342033  -6.73560975
  -6.70795048  -6.68044142  -6.65308145  -6.62585078  -6.59874877
```

```
-6.57179303  -6.5449825   -6.51831611  -6.49179281  -6.46541156
-6.43917134  -6.41307111  -6.38710986  -6.36128659  -6.33556241
-6.30996353  -6.28450023  -6.25917154  -6.23397649  -6.20891412
-6.18398348  -6.15918365  -6.13451368  -6.10997266  -6.08553627
-6.06119495  -6.03698048  -6.01289195  -5.98892848  -5.96508919
-5.9413732   -5.91777966  -5.89430769  -5.87095646  -5.84772512
-5.82461284  -5.80161798  -5.77871174  -5.75592242  -5.73324921
-5.71069132  -5.68824794  -5.6659183   -5.6437016   -5.62159707
-5.59960394  -5.57772146  -5.55594886  -5.53425415  -5.51266035
-5.49117463]
```

In [105…

```
# Gerando a função valor e a função política dos agentes que já receberam herança
V_after = np.zeros((a_size, N, 9))
g_after = np.zeros((a_size, N, 9))

# Preenchendo o ultimo período (que é dado em função de V_ret)
for i in range(N):
    for j in range(a_size):
        estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
        if j == 0:
            start = copy.deepcopy(0)
        valor = np.zeros((1,a_size))
        c_t = (estoque - grid_a[start])
        if c_t >= 0:
            u = ((c_t**(1-mu) - 1)/(1-mu))
        elif c_t < 0:
            u = -1/(1-mu)
        valor[:,(start)] = u + beta*V_ret[(start),0]
        for k in range((start+1), a_size):
            c_k = (estoque - grid_a[k])
            if c_k >= 0:
                u = ((c_k**(1-mu) - 1)/(1-mu))
            elif c_k < 0:
                u = -1/(1-mu)
            valor[:,k] = u + beta*(surv[9]*V_ret[k,0]+(1-surv[9])*psi(k))
            if valor[:,k]< valor[:,k-1]:
                break
        if valor[:,k]< valor[:,k-1]:
            V_after[j,i,8] = copy.deepcopy(valor[:,k-1])
            indice_a = copy.deepcopy(k-1)
        elif valor[:,k] >= valor[:,k-1]:
            V_after[j,i,8] = copy.deepcopy(valor[:,k])
            indice_a = copy.deepcopy(k)
        start = copy.deepcopy(indice_a)
        g_after[j,i,8] = copy.deepcopy(grid_a[indice_a])

for t in range(1,9):
    for i in range(N):
        probabilidades = b[i,:]
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = copy.deepcopy(0)
            valor = np.zeros((1,a_size))
            c_t = (estoque - grid_a[start])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca = 0
            for w in range(N):
                esperanca += probabilidades[w]*V_after[(start),w,(9-t)]
            valor[:,(start)] = u + beta*esperanca
```

```
            for k in range((start+1), a_size):
                c_k = (estoque - grid_a[k])
                if c_k >= 0:
                    u = ((c_k**(1-mu) - 1)/(1-mu))
                elif c_k < 0:
                    u = -1/(1-mu)
                esperanca = 0
                for w in range(N):
                    esperanca += probabilidades[w]*V_after[k,w,(9-t)]
                valor[:,k] = u + beta*esperanca
                if valor[:,k]< valor[:,k-1]:
                    break
            if valor[:,k]< valor[:,k-1]:
                V_after[j,i,(8-t)] = copy.deepcopy(valor[:,k-1])
                indice_a = copy.deepcopy(k-1)
            elif valor[:,k] >= valor[:,k-1]:
                V_after[j,i,(8-t)] = copy.deepcopy(valor[:,k])
                indice_a = copy.deepcopy(k)
            start = copy.deepcopy(indice_a)
            g_after[j,i,(8-t)] = copy.deepcopy(grid_a[indice_a])
```

In [106…

```
# Desejamos a distribuição de ativos de acordo com a idade
dist_et_a = np.zeros((a_size,14))

# todo agente nasce com zero ativos, portanto
dist_et_a[40,0] = 1

# Desse ponto em diante, precisaremos estimar uma distribuição inical dos ativos, o
# distribuição uniforme nos ativos para os demais perídos até sua aposentadoria.
# depois, utilizaremos a função política dos aposentados para obter uma estimativa i
for t in range(1,9):
    for j in range(a_size):
        dist_et_a[j,t]=dist_a[j]

for t in range(9,14):
    aux = np.zeros(a_size)
    for k in range(a_size):
        for j in range(a_size):
            if a_ret[k,(t-9)]==grid_a[j]:
                aux[j]+=dist_et_a[k,(t-1)]
    for j in range(a_size):
        dist_et_a[j,t] = aux[j]
```

In [107…

```
# Primeiro, vamos criar uma distribuição invariante nos estados, pois isso será impo
# conjunta dos estados e ativos para os agentes que acabaram de nascer
dist_0 = np.full( N,(1/N))
tol_distrb = 10**(-6)
dist_old = copy.deepcopy(dist_0)
dist_s = copy.deepcopy(dist_0)
dist_diff = np.ones(N)
erro=1
while(erro>tol_distrb):
    t += 1
    for i in range(N):
        dist_s[i] = dist_old@b[:,i]
        dist_diff[i] = (dist_s[i]-dist_old[i])
        dist_diff[i] = np.abs(dist_diff[i])
    erro = np.max(dist_diff)
    dist_old = copy.deepcopy(dist_s)
```

```
# Criando a distribuição conjunta inicial
pi_0 = np.zeros((a_size, N))
a_zero = np.where(grid_a==0)
for i in range(N):
    pi_0[a_zero, i] = dist_s[i]
```

In [108…

```
# Gerando a função valor e a função política dos agentes que ainda não receberam her
V_before = np.zeros((a_size, N, 9))
g_before = np.zeros((a_size, N, 9))
pi_old = copy.deepcopy(pi_0)
pi_new = copy.deepcopy(pi_0)
dist_et_a2 = copy.deepcopy(dist_et_a)
tol = 10**(-2)
erro = 1
cont = 0
# Preenchendo o ultimo período (que é dado em função de V_ret)
while erro>=tol:
    print(cont)
    cont += 1
    dist_et_a = copy.deepcopy(dist_et_a2)
    for i in range(N):
        for j in range(a_size):
            estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
            if j == 0:
                start = copy.deepcopy(0)
            valor = np.zeros((1,a_size))
            c_t = (estoque - grid_a[start])
            if c_t >= 0:
                u = ((c_t**(1-mu) - 1)/(1-mu))
            elif c_t < 0:
                u = -1/(1-mu)
            esperanca2 = 0
            for w in range(a_size):
                esperanca2 += dist_et_a[w,13]*V_ret[max(0,min((start+w),(a_size-1)))
            valor[:,(start)] = u + beta*(surv[13]*V_ret[(start),0]+(1-surv[13])*espe
            for k in range((start+1), a_size):
                c_k = (estoque - grid_a[k])
                if c_k >= 0:
                    u = ((c_k**(1-mu) - 1)/(1-mu))
                elif c_k < 0:
                    u = -1/(1-mu)
                esperanca2 = 0
                for w in range(a_size):
                    esperanca2 += dist_et_a[w,13]*V_ret[max(0,min((k+w),(a_size-1)))
                valor[:,k] = u + beta*(surv[13]*V_ret[k,0]+(1-surv[13])*esperanca2)
                if valor[:,k]< valor[:,k-1]:
                    break
            if valor[:,k]< valor[:,k-1]:
                V_before[j,i,8] = copy.deepcopy(valor[:,k-1])
                indice_a = copy.deepcopy(k-1)
            elif valor[:,k] >= valor[:,k-1]:
                V_before[j,i,8] = copy.deepcopy(valor[:,k])
                indice_a = copy.deepcopy(k)
            start = copy.deepcopy(indice_a)
            g_before[j,i,8] = copy.deepcopy(grid_a[indice_a])


    for t in range(1,9):
        for i in range(N):
            probabilidades = b[i,:]
            for j in range(a_size):
                estoque = np.exp(z_grid[i]) + (1+r)*grid_a[j]
```

```
                if j == 0:
                    start = copy.deepcopy(0)
                valor = np.zeros((1,a_size))
                c_t = (estoque - grid_a[start])
                if c_t >= 0:
                    u = ((c_t**(1-mu) - 1)/(1-mu))
                elif c_t < 0:
                    u = -1/(1-mu)
                esperanca1 = 0
                for w in range(N):
                    esperanca1 += probabilidades[w]*V_after[(start),w,(9-t)]
                esperanca2 = 0
                for w in range(N):
                    for v in range(a_size):
                        esperanca2 += probabilidades[w]*dist_et_a[v,13-t]*V_after[ma
                valor[:,(start)] = u + beta*(surv[13-t]*esperanca1+(1-surv[13-t])*es
                for k in range((start+1), a_size):
                    c_k = (estoque - grid_a[k])
                    if c_k >= 0:
                        u = ((c_k**(1-mu) - 1)/(1-mu))
                    elif c_k < 0:
                        u = -1/(1-mu)
                    esperanca1 = 0
                    for w in range(N):
                        esperanca1 += probabilidades[w]*V_after[k,w,(9-t)]
                    esperanca2 = 0
                    for w in range(N):
                        for v in range(a_size):
                            esperanca2 += probabilidades[w]*dist_et_a[v,13-t]*V_afte
                    valor[:,k] = u + beta*(surv[13-t]*esperanca1+(1-surv[13-t])*espe
                    if valor[:,k]< valor[:,k-1]:
                        break
                if valor[:,k]< valor[:,k-1]:
                    V_before[j,i,(8-t)] = copy.deepcopy(valor[:,k-1])
                    indice_a = copy.deepcopy(k-1)
                elif valor[:,k] >= valor[:,k-1]:
                    V_before[j,i,(8-t)] = copy.deepcopy(valor[:,k])
                    indice_a = copy.deepcopy(k)
                start = copy.deepcopy(indice_a)
                g_before[j,i,(8-t)] = copy.deepcopy(grid_a[indice_a])

    for t in range(9):
        g_beforeiter = copy.deepcopy(g_before[:,:,t])
        g_afteriter = copy.deepcopy(g_after[:,:,t])
        for i in range(N):
            for j in range(a_size):
                aux1 = pi_old*(g_beforeiter == grid_a[j])
                aux2 = pi_old*(g_afteriter == grid_a[j])
                pi_new[j,i] = surv[t+5]*sum(aux1)@b[:,i] + (1 -surv[t+5])*sum(aux2)@
        d = np.zeros(a_size)

        for j in range(a_size):
            for i in range(N):
                d[j]+=pi_new[j,i]
        dist_et_a2[:,t+1] =copy.deepcopy(d)
        pi_old = copy.deepcopy(pi_new)
    for t in range(9,14):
        aux = np.zeros(a_size)
        for k in range(a_size):
            for j in range(a_size):
                if a_ret[k,(t-9)]==grid_a[j]:
                    aux[j]+=dist_et_a2[k,(t-1)]
        for j in range(a_size):
            dist_et_a2[j,t] = aux[j]
```

```
    dif = np.abs(dist_et_a2 - dist_et_a)
    erro = np.max(np.max(dif))
    print(erro)
```

```
0
0.4153146301144736
1
0.4741148565611896
2
0.23043302520902806
3
0.6587135118469519
4
0.10249090346034724
5
0.3520356425389905
6
0.27072205532069044
7
0.5854950793234475
8
0.5940337901668192
9
0.3323163462722032
10
0.25843652166494313
11
0.39430350607328757
12
0.5131068383380989
13
0.26367873934662367
14
0.2452571553416677
15
0.5753769136129688
16
0.5756896714466606
17
0.3315633982056648
18
0.3112022797745657
19
0.07548837580123596
20
0.2637053275597126
21
0.15759801014379624
22
0.4504110825399611
23
0.35536068465986465
24
0.21571158772643623
25
0.20502490700583223
26
0.38753860892565795
27
0.3023579003196012
28
0.34071487550491747
```

```
29
0.43894116151023477
30
0.34550011597247043
31
0.23969653287994724
32
0.40061452628880695
33
0.3074147302362128
34
0.2693548804138938
35
0.4390491746222332
36
0.3443310909874821
37
0.2518959217271941
38
0.38801126408976333
39
0.3030158630181039
40
0.42547586140643806
41
0.43956214478649197
42
0.34209874940653184
43
0.23699112261340208
44
0.37391039293749967
45
0.29950978029022396
46
0.4213086388817817
47
0.4403771533090069
48
0.33938137666946133
49
0.2377178263276408
50
0.34289966026948887
51
0.28282316495466586
52
0.36186339272122225
53
0.4431805678684899
54
0.3309916031481091
55
0.23756558113510884
56
0.23419634117386218
58
0.40506796847529236
59
0.4522068200908891
60
0.2991608401023506
61
0.22718472566665596
```

```
62
0.22748247159410748
63
0.24904957830278893
64
0.2490866980567889
65
0.1877998530470989
66
0.13655349005310036
67
0.15393051106251343
68
0.3920243801409344
69
0.4323257658898317
70
0.34462828208840135
71
0.24064530099682344
72
0.38750710905891095
73
0.31318313481317006
74
0.4017682822564128
75
0.44041053538677155
76
0.3413822685082242
77
0.2352782814687766
78
0.353436114496549
79
0.28658431939220436
80
0.37607550421459024
81
0.4416500805568585
82
0.33481246117663066
83
0.23654570062679545
84
0.30205824763356437
85
0.2548685215108295
86
0.23930459689009503
87
0.4479088623143248
88
0.31257743111368275
89
0.23245218657354857
90
0.2407080200686196
91
0.22505218346014225
92
0.4226950995706435
93
0.47954692814938277
```

```
94
0.2764042691247577
95
0.26909519478795907
96
0.08237713436846639
97
0.22565337553966747
98
0.3503932288732092
99
0.3134005106129917
100
0.21887077425037732
101
0.23451856846542452
102
0.22371329956583608
103
0.37912052985665373
104
0.47290091590880695
105
0.2708417450614984
106
0.265118435527144
107
0.08243815291494158
108
0.28513726831803066
109
0.3496924530502534
110
0.3383053385065095
111
0.23602920182806852
112
0.30635909419021146
113
0.2563600816751186
114
0.2508671484026208
115
0.44651183342540524
116
0.3145761974030931
117
0.23788018900443042
118
0.24284370895096052
119
0.220936127828508
120
0.40422702516545805
121
0.4989921507368794
122
0.31517107692696955
123
0.29580552881643885
124
0.08009170747809809
125
0.24669468575957756
```

```
126
0.22343807804103835
127
0.333344601250346
128
0.347626531774326
129
0.28466372271474427
130
0.36602417107793717
131
0.29677478857896705
132
0.443116742749956
133
0.443375153748923
134
0.33671455303674475
135
0.2386347804218054
136
0.31070996867486256
137
0.26056359810677643
138
0.2883159891026029
139
0.44661181355584995
140
0.3180044734564095
141
0.23968370109145826
142
0.24529407521230537
143
0.2214794835367345
144
0.27618624424816024
145
0.4903073154016374
146
0.30433081567528475
147
0.285193264923714
148
0.09051067301336212
149
0.2424353739887646
150
0.1947676181274321
151
0.324966036231008
152
0.3489861266968761
153
0.29265163435112757
154
0.4087209576471618
155
0.3241701181634398
156
0.3252862509861065
157
0.43979362889575
```

```
158
0.3426107397848984
159
0.2610606224500652
160
0.3827335326257415
161
0.3051027575151876
162
0.43040979429255116
163
0.43993565793798217
164
0.34098806096183226
165
0.23557185478873097
166
0.35347207408052006
167
0.2865803198213511
168
0.3760754661019296
169
0.4416500829950797
170
0.33481246028517336
171
0.23654570060644292
172
0.30205824763394507
173
0.25486852150968575
174
0.23930459689007721
175
0.44790886231431315
176
0.3125774311136742
177
0.23245218657354255
178
0.24070802006861336
179
0.2250521834601364
180
0.4226950995706322
181
0.47954692814936983
182
0.27640426912475025
183
0.2690951947879519
184
0.0823771343684642
185
0.22565337553966147
186
0.3503932288732
187
0.3134005106129834
188
0.21887077425037155
189
0.23451856846541816
```

```
190
0.22371329956583003
191
0.37912052985664363
192
0.47290091590879435
193
0.27084174506149106
194
0.26511843552713676
195
0.08243815291493942
196
0.2851372683180231
197
0.3496924530502442
198
0.3383053385065006
199
0.23602920182806225
200
0.3063590941902033
201
0.2563600816751117
202
0.25086714840261437
203
0.44651183342539336
204
0.31457619740308473
205
0.23788018900442406
206
0.24284370895095406
207
0.22093612782850208
208
0.40422702516544745
209
0.49899215073686615
210
0.3151710769269611
211
0.2958055288164309
212
0.08009170747809596
213
0.2466946857595709
214
0.22343807804103238
215
0.33334460125033705
216
0.3476265317743168
217
0.2846637227147366
218
0.3660241710779272
219
0.296774788578959
220
0.44311674274994417
221
0.4433751537489112
```

```
222
0.3367145530367353
223
0.23863478042179884
224
0.31070996867485406
225
0.26056359810676927
226
0.28831598910259515
227
0.4466118135558377
228
0.3180044734564007
229
0.23968370109145162
230
0.24529407521229862
231
0.22147948353672842
232
0.27618624424815263
233
0.4903073154016239
234
0.3043308156752763
235
0.28519326492370634
236
0.09051067301335954
237
0.242435373988758
238
0.19476761812742674
239
0.32496603623099896
240
0.3489861266968664
241
0.29265163435111935
242
0.4087209576471507
243
0.324170118163431
244
0.3252862509860975
245
0.4397936288957379
246
0.342610739784889
247
0.26106062245005807
248
0.382733532625731
249
0.30510275751517923
250
0.4304097942925392
251
0.4399356579379702
252
0.34098806096182294
253
0.23557185478872453
```

```
254
0.3534720740805103
255
0.28658031982134324
256
0.3760754661019191
257
0.44165008299506736
258
0.334812460285164
259
0.2365457006064364
260
0.30205824763393657
261
0.2548685215096786
262
0.2393045968900707
263
0.4479088623143008
264
0.31257743111366565
265
0.232452186573536
266
0.24070802006860667
267
0.22505218346013015
268
0.4226950995706206
269
0.4795469281493568
270
0.2764042691247427
271
0.26909519478794464
272
0.08237713436846206
273
0.22565337553965534
274
0.3503932288731904
275
0.3134005106129749
276
0.21887077425036547
277
0.2345185684654118
278
0.22371329956582395
279
0.3791205298566332
280
0.47290091590878125
281
0.27084174506148356
282
0.26511843552712944
283
0.08243815291493713
284
0.28513726831801517
285
0.3496924530502342
```

```
286
0.33830533850649097
287
0.23602920182805567
288
0.30635909419019486
289
0.2563600816751047
290
0.2508671484026072
291
0.4465118334253808
292
0.31457619740307585
293
0.23788018900441746
294
0.24284370895094737
295
0.220936127828496
296
0.4042270251654361
297
0.4989921507368521
298
0.31517107692695223
299
0.29580552881642264
300
0.08009170747809374
301
0.24669468575956413
302
0.22343807804102617
303
0.33334460125032783
304
0.34762653177430713
305
0.2846637227147288
306
0.366024171077917
307
0.29677478857895073
308
0.4431167427499318
309
0.44337515374889874
310
0.33671455303672615
311
0.23863478042179234
312
0.3107099686748454
313
0.260563598106762
314
0.28831598910258704
315
0.4466118135558253
316
0.3180044734563918
317
0.2396837010914451
```

```
318
0.24529407521229185
319
0.22147948353672228
320
0.27618624424814503
321
0.4903073154016103
322
0.3043308156752679
323
0.2851932649236983
324
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_2776/2972832650.py in <module>
     57                 if j == 0:
     58                     start = copy.deepcopy(0)
---> 59                 valor = np.zeros((1,a_size))
     60                 c_t = (estoque - grid_a[start])
     61                 if c_t >= 0:

KeyboardInterrupt:
```

In [ ]:

```
    for t in range(9):
        g_beforeiter = copy.deepcopy(g_before[:,:,t])
        for i in range(N):
            for j in range(a_size):
                aux = pi_old*(g_beforeiter == grid_a[j])
                pi_new[j,i] = sum(aux)@b[:,i]
        d = np.zeros(a_size)

        for j in range(a_size):
            for i in range(N):
                d[j]+=pi_new[j,i]
        dist_et_a[:,t+1] =copy.deepcopy(d)
        pi_old = copy.deepcopy(pi_new)
    for t in range(9,14):
        aux = np.zeros(a_size)
        for k in range(a_size):
            for j in range(a_size):
                if a_ret[k,(t-9)]==grid_a[j]:
                    aux[j]+=dist_et_a[k,(t-1)]
        for j in range(a_size):
            dist_et_a[j,t] = aux[j]
```

In [ ]:

```
print(g_before[:,4,8])
print(g_after[:,4,8])
```

In [ ]:

```
a_size = 201
test_a = np.full((a_size,14),1/a_size)
test_b = copy.deepcopy(test_a)

test_b[87,9] = 1.28/a_size
test_b[32,9] += -0.8*test_b[87,9]
test_b[91,9] += -0.2*test_b[87,9]

test_dif = np.abs(test_b - test_a)
print(test_dif[:,9])
```

```
test_erro = np.max(np.max(test_dif))
print(test_erro)
```

In [ ]: