Problem1_2_Jeanne

November 15, 2020

The code and write up below correspond to questions 1 and 2 of the computational assignment. The code is set up such that all functions are defined at the beginning, and ran later in the problem. Notice that I do not use the transition matrix T for problem 1. After implementing it for problem 2 I aimed at rewritting the code for problem 1 to use the (time saving) approach of T, but I ran out of time. After understanding the Transition matrix approach through spending a positive number of hours debugging the transition matrix function, I recognize it is indeed a very convenient way to set the problem up, which I didn't know about.

$v1$ refers to question 1, $v2$ refers to question 2.

[1]:
```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.sparse import csr_matrix
import time
from scipy import sparse
import numpy.linalg as lin
```

[2]:
```python
# Define primitives
def u_f1(S,y):
    '''Utility function version 1'''
    U = 2*y**0.5
    if y > S:
        U = -100000000
    return U

def u_f2(S,y):
    '''Utility function version 2'''
    U = 5*y - 0.05*y**2
    if y > S:
        U = -100000000
    return U

def Bellman_simple(Vp, ik, jk, utilitymatrix):
    '''
    Bellman Function for the non-interpolation case (Problem 1)
    '''
    Value = utilitymatrix[ik, jk] +   * Vp[ik-jk]
    return(Value)
```

```
def Bellman_interp(Vp, ik, jk, utilitymatrix, T):
    '''
    Bellman : value function for state variable Kgrid[ik] with policy Kgrid[jk]␣
 ↪(Problem 2)
    '''
    N = len(T)
    beg = N*jk
    fin = N*(jk+1)
    subT = T[ik, beg:fin] # get the index of the new state by subsetting in the␣
 ↪transition matrix
    state_new = np.where(subT > 0)[0]
    if len(state_new)==1: # Reach the boundary: extraction > stock --> next␣
 ↪stock == 0
        state_new1 = state_new[0]
        weight1 = subT[state_new1]
        Vnext = Vp[state_new1]*weight1
    if len(state_new)>1:
        state_new1 = state_new[0]
        state_new2 = state_new[1]
        weight1 = subT[state_new1]
        weight2 = subT[state_new2]
        Vnext = Vp[state_new1]*weight1 + Vp[state_new2]*weight2
    Value = utilitymatrix[ik, jk]  +   * Vnext
    return (Value)

# Price functions = derivatives of utility functions wrt extraction (MU)
def p1_f(C1):
    return (1/(C1**0.5))

def p2_f(C2):
    return (5 - 0.1*C2)
```

[3]:

```
def T_f(S, A, interp=True):
    '''
    This function computes the transition matrix for the interpolation problem␣
 ↪(Problem 2)
    It can be easily adapted to the standard case
    '''

    # Initialize
    N = len(S)
    Na = len(A)
    T = np.zeros((N, N*Na))

    # For each action indexed k, and each initial state i, compute resulting␣
 ↪state j
```

```
    if interp==True:
        for i in range(N):
            for k in range(Na):
                diff = S[i] - A[k]
                if diff < 0:
                    index1_raw = 0
                    index1_adj = index1_raw + (N)*k
                    T[i, index1_adj] = 1.0
                if diff >= 0:
                    dist = abs(S - diff) # How far from the difference is each␣
↪element of S
                    index1_raw = np.where(dist == sorted(dist)[0])[0][0] # Find␣
↪the index of the closest
                    index2_raw = np.where(dist == sorted(dist)[1])[0][0] # Find␣
↪the index of the second closest
                    weight1 = 1 - sorted(dist)[0] / np.sum(sorted(dist)[0:2])
                    weight2 = 1 - sorted(dist)[1] / np.sum(sorted(dist)[0:2])
                    index1_adj = index1_raw + (N)*k
                    index2_adj = index2_raw + (N)*k
                    T[i, index1_adj] = weight1
                    T[i, index2_adj] = weight2
    return(T)
```

[4]:
```
def VFI_v1(State_grid, Action_grid, V0, utilitymatrix, maxiter = 1000,␣
↪tol=1e-8,  =(1/1.05), howard=True, print_i=True):
    '''
    VFI routine for problem 1
    (should be merged with VFI_v2 for concision but I'm running out of time)
    '''
    N = len(State_grid)
    Na = len(Action_grid)
    ### Initiate vectors & numbers
    iter = 0
    epsi = 1
    IndexAction_new = np.zeros(Na)
    Action = np.zeros(Na)
    Vp = V0
    Vp_new = np.zeros(N)

    while (epsi > tol) & (iter < maxiter):
        for ik in range(N):
            # ik = index of the stock today
            # d = max extraction < current stock
            #d = int(gk[max(ik-1), 0])
            d = 0

            #Value function Tj for each choice g(k)
            Vnext = np.zeros(len(State_grid)-d)

            # Compute the value function Tj for each choice of g(k)
            for jk in range(d, Na):
                    Vnext[jk - d] = Bellman_simple(Vp, ik, jk, utilitymatrix)

            # Choose the maximum
            Vp_new[ik] = np.max(Vnext)
            #IndexAction_new[ik] = d + int(np.where(Vnext==np.max(Vnext))[0][0])
            IndexAction_new[ik] = d + int(np.max(np.where(Vnext==np.
↪max(Vnext))[0]))


        if howard==True:
            # Reduces the number of iterations by updating the Vnew grid more␣
↪often
            for c in range(100):
                for ik in range(len(Action_grid)):
                    jk = int(IndexAction_new[ik])
                    Vp_new[ik] = Bellman_simple(Vp_new, ik, jk, utilitymatrix)


        ### Check the error
        epsi = np.abs(Vp_new - Vp).max()

        ### Keep track of what is going on
        if print_i==True:
            print("iteration is ", iter, " and Error term is ", epsi)

        IndexAction = IndexAction_new*1. #Update
        Vp = Vp_new*1. #Update

        if epsi < tol:
            break

        iter=iter+1

    #Calculate Values for g(k)
    for jk in range(len(Action_grid)):
        Action[jk] = Action_grid[int(IndexAction[jk])]
    print("Total number of iterations : ", iter)

    # Calculate the N*N optimal transition matrix T_opt that gives the state␣
↪transition proba under optimal action
    return[Vp, IndexAction, Action, iter]
```

```
[5]: def VFI_v2(State_grid, Action_grid, V0, utilitymatrix, T, maxiter = 1000, tol =␣
     ↪1e-8,  =(1/1.05), print_i=True):
         '''
         VFI Routine for Problem 2
         '''

         ### Initiate vectors & numbers
         iter = 0
         epsi = 1
         N = len(State_grid)
         Na = len(Action_grid)
         IndexAction_new = np.zeros(Na)
         Vp = np.zeros(N)

         while (epsi > tol) & (iter < maxiter):
             # Copy the old value vector
             Vold = np.copy(Vp)
             # Initiate the matrix
             Vnextall = np.zeros((N, Na))
             # For each Na*N block (each action): computes the pair [old, new] states
             for jk in range(Na):
                 beg = N*jk
                 fin = N*(jk+1)
                 Tnext = T[:,beg:fin]
                 vals = T[:,beg:fin].dot(Vp)
                 Vnextall[:,jk] = vals
             # V = U +   V
             Uall = utilitymatrix +   * Vnextall
             # Find the maximum total V for each starting i (Vp[i]), and the index␣
     ↪of the corresponding action maxArgAction[i]
             maxArgAction = np.zeros(N)
             for i in range(N):
                 maxArgAction[i] = np.argmax(Uall[i,:])
                 Vp[i] = np.max(Uall[i,:])

             ### Check the error
             epsi = lin.norm(Vp - Vold)

             ### Keep track of what is going on
             if print_i==True:
                 print("iteration is ", iter, " and Error term is ", epsi)

             IndexAction_new = np.copy(maxArgAction)

             if epsi < tol:
                 break
```

```
        iter=iter+1

    return(Vp, IndexAction_new, iter)
```

0.1 Question 1: Solving the DDP by value function iteration ; discrete state space

[6]:
```
### Parameters
r = 0.05
  = 1/(1+r)
Stot = 1000

N = 501
Na = 501
S1 = np.linspace(0,1000,N)
A1 = np.linspace(0,1000,Na)
V0 = np.zeros(N)

### Compute utility matrix for utility 1 and 2
utilitymatrix1_v1 = np.ones((N, Na))
for i in range(0,N):
    for j in range(0,Na):
        utilitymatrix1_v1[i,j] = u_f1(S1[i], A1[j])

utilitymatrix2_v1 = np.ones((N, Na))
for i in range(0,N):
    for j in range(0,Na):
        utilitymatrix2_v1[i,j] = u_f2(S1[i], A1[j])
```

[7]:
```
##### UTILITY 1
### Solve the model
[Vp1_v1, IndexAction1_v1, Action1_v1, iter1_v1] = VFI_v1(S1, A1, V0,␣
 ↪utilitymatrix1_v1, howard=True, print_i=False)
print("solving model 1 done")

### Calculate the N*N optimal transition matrix
T_opt1 = np.zeros((N, N))
for ik in range(N):
    ij = np.where(S1 == S1[ik] - Action1_v1[ik])[0][0]
    T_opt1[ik, ij] = 1

##### UTILITY 2
### Solve the model
[Vp2_v1, IndexAction2_v1, Action2_v1, iter2_v1] = VFI_v1(S1, A1, V0,␣
 ↪utilitymatrix2_v1, howard=True, print_i=False)
```

```
print("solving model 2 done")

### Calculate the N*N optimal transition matrix
T_opt2 = np.zeros((N, N))
for ik in range(N):
    ij = np.where(S1 == S1[ik] - Action2_v1[ik])[0][0]
    T_opt2[ik, ij] = 1
```

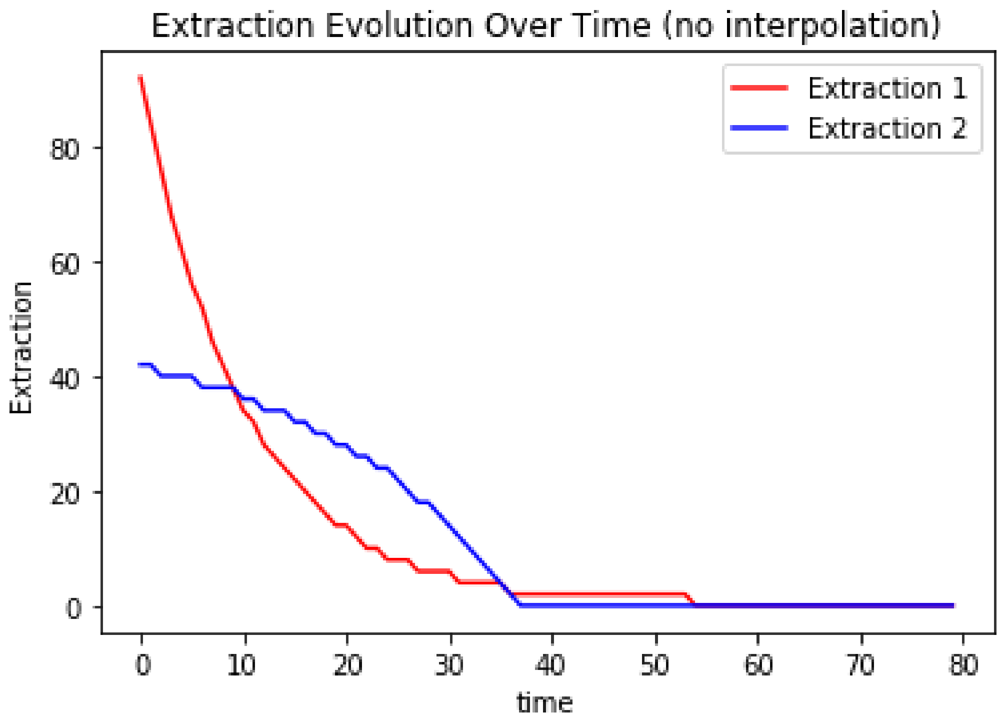
```
Total number of iterations :  22
solving model 1 done
Total number of iterations :  14
solving model 2 done
```

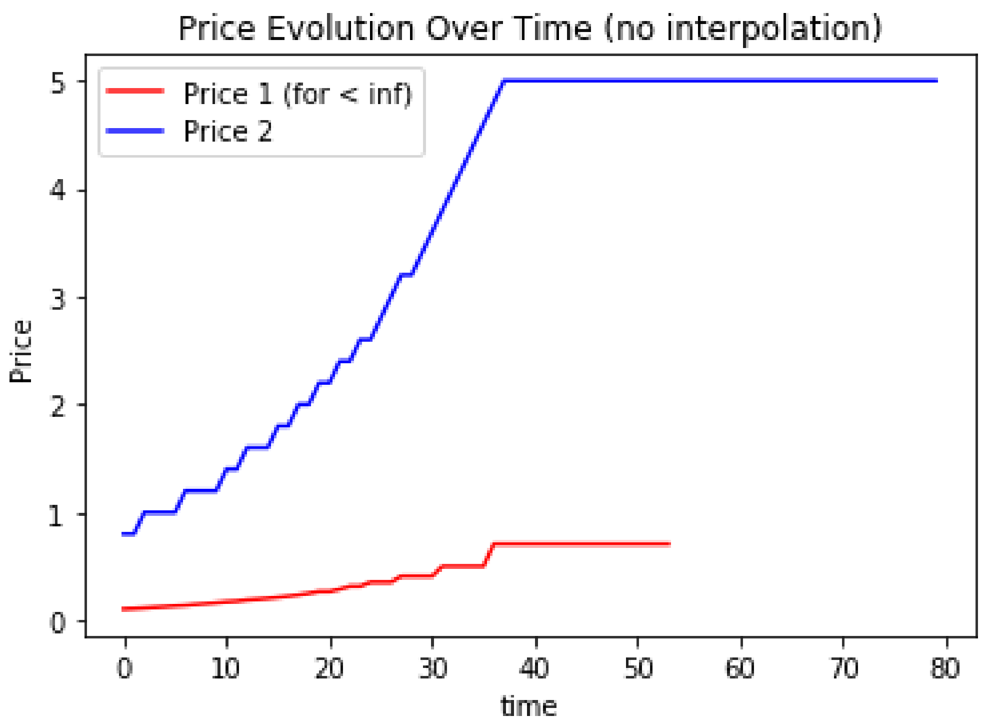
[8]:
```
##### Simulate the model for 80 periods
# St = each period's stock
# C = each period's extraction

## Simulate 1
St1_v1 = 1000*np.ones(80)
C1_v1 = np.zeros(80)
for i in range(80-1):
    init_index1 = np.where(S1==St1_v1[i])[0][0]
    C1_v1[i] = Action1_v1[init_index1]
    final_index1 = np.where(T_opt1[init_index1,]==1)[0][0]
    St1_v1[i+1] = S1[final_index1]
p1_v1 = p1_f(C1_v1)

## Simulate 2
St2_v1 = 1000*np.ones(80)
C2_v1 = np.zeros(80)
for i in range(80-1):
    init_index2 = np.where(S1==St2_v1[i])[0][0]
    C2_v1[i] = Action2_v1[init_index2]
    final_index2 = np.where(T_opt2[init_index2,]==1)[0][0]
    St2_v1[i+1] = S1[final_index2]
p2_v1 = p2_f(C2_v1)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:47:
RuntimeWarning: divide by zero encountered in true_divide
```

[9]:
```
plt.plot(range(0,80), C1_v1, color="red", label="Extraction 1")
plt.plot(range(0,80), C2_v1, color="blue", label="Extraction 2")
plt.title("Extraction Evolution Over Time (no interpolation)")
plt.xlabel("time")
plt.ylabel("Extraction")
plt.legend()
plt.show()
```

```
# For printing purposes
plt.plot(range(0,80), p1_v1, color="red", label="Price 1 (for < inf)")
plt.plot(range(0,80), p2_v1, color="blue", label="Price 2")
plt.title("Price Evolution Over Time (no interpolation)")
plt.xlabel("time")
plt.ylabel("Price")
plt.legend()
plt.show()
```

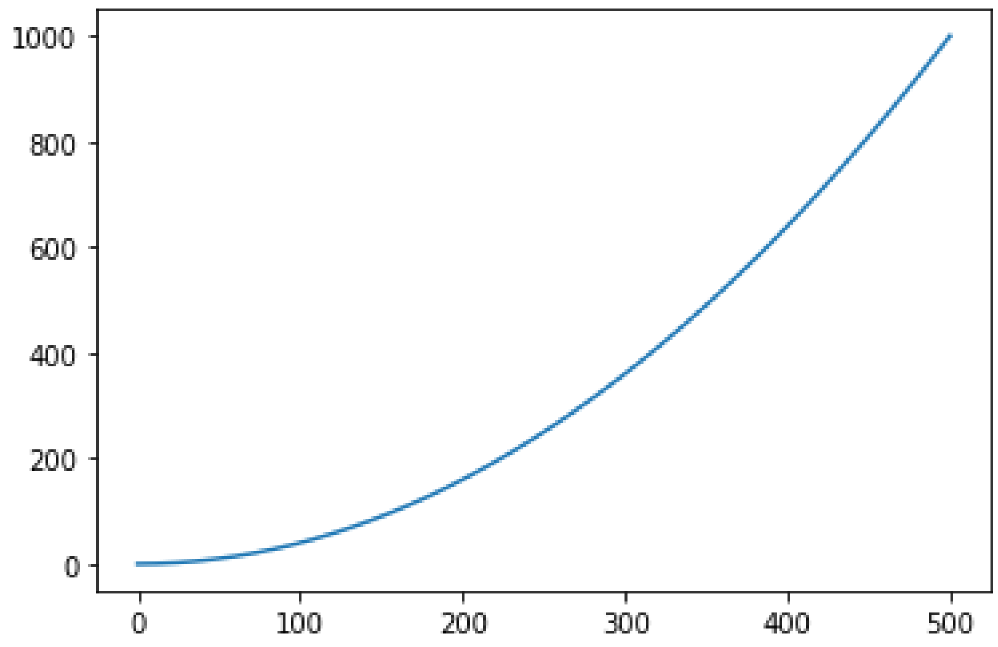




0.2 Question 2: Interpolating Between the States

```
[10]: N2 = 501
      Na2 = 501
      S2 = np.linspace(0,1000,N2)
      A2 = np.linspace(0,(1000**0.5),Na2)**2
      plt.plot(range(501), A2)
```

```
[10]: [<matplotlib.lines.Line2D at 0x7fab364f9a90>]
```



[11]:
```
### Compute utility matrix for utility 1 and 2
utilitymatrix1_v2 = np.ones((N2, Na2))
for i in range(0,N2):
    for j in range(0,Na2):
        utilitymatrix1_v2[i,j] = u_f1(S2[i], A2[j])

utilitymatrix2_v2 = np.ones((N2, Na2))
for i in range(0,N2):
    for j in range(0,Na2):
        utilitymatrix2_v2[i,j] = u_f2(S2[i], A2[j])
```

[12]:
```
### Compute the Transition Matrix
T2 = T_f(S2, A2, interp=True)
T2 = csr_matrix(T2)
```

[13]:
```
### VFI
V0 = np.zeros(N2)
[Vp1_v2, IndexAction1_v2, iter1_v2] = VFI_v2(S2, A2, V0, utilitymatrix1_v2, T =␣
 ↪T2, print_i=True)
print("Utility 1: DONE")

[Vp2_v2, IndexAction2_v2, iter2_v2] = VFI_v2(S2, A2, V0, utilitymatrix2_v2, T =␣
 ↪T2, print_i=False)
print("Utility 2: DONE")
```

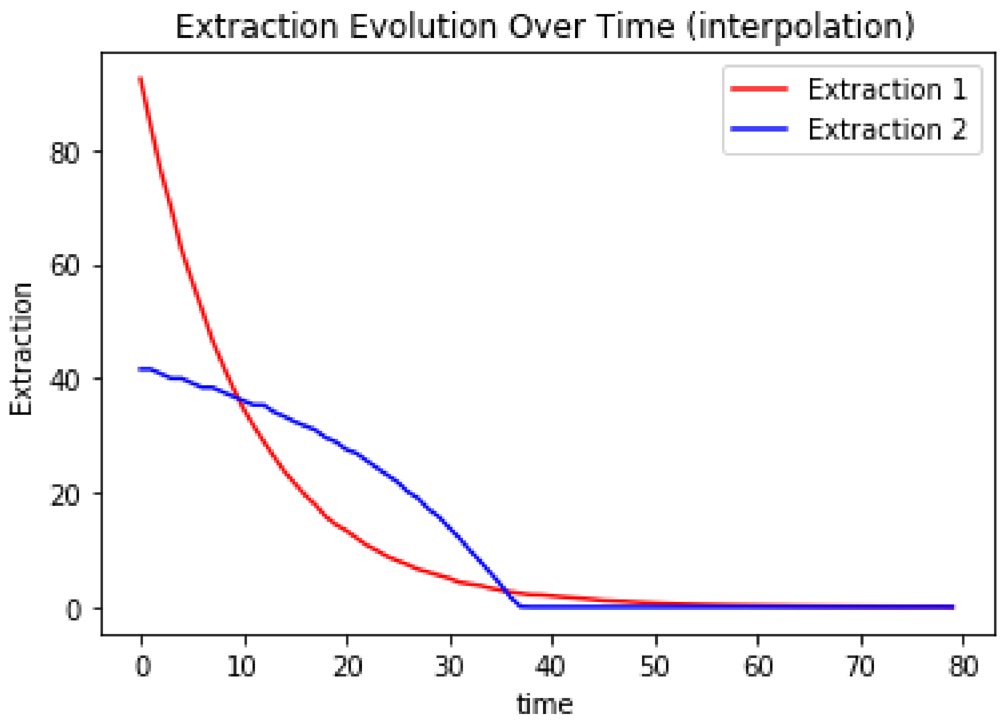
```
iteration is  0  and Error term is  999.6306757998175
iteration is  1  and Error term is  382.22358359005364
iteration is  2  and Error term is  271.571086320296
iteration is  3  and Error term is  212.4579473176012
iteration is  4  and Error term is  173.61935439472015
iteration is  5  and Error term is  145.52413508394983
iteration is  6  and Error term is  124.15091213179542
iteration is  7  and Error term is  107.05369166338109
iteration is  8  and Error term is  93.12997699892438
iteration is  9  and Error term is  81.61902093185495
iteration is  10  and Error term is  71.86880582805578
iteration is  11  and Error term is  63.499016900233514
iteration is  12  and Error term is  56.34387846166145
iteration is  13  and Error term is  50.11094282974448
iteration is  14  and Error term is  44.7271788902159
iteration is  15  and Error term is  40.03579837182327
iteration is  16  and Error term is  35.830256185787505
iteration is  17  and Error term is  32.08515277453874
iteration is  18  and Error term is  28.793217288378887
iteration is  19  and Error term is  25.85500434718936
iteration is  20  and Error term is  23.263422138973258
iteration is  21  and Error term is  20.937125947878076
iteration is  22  and Error term is  18.864180362648234
iteration is  23  and Error term is  16.992165851380783
iteration is  24  and Error term is  15.301786992192639
iteration is  25  and Error term is  13.796044899269988
iteration is  26  and Error term is  12.462800808911043
iteration is  27  and Error term is  11.278960520867294
iteration is  28  and Error term is  10.208583906133356
iteration is  29  and Error term is  9.222334728356675
iteration is  30  and Error term is  8.330500392772214
iteration is  31  and Error term is  7.524855551341117
iteration is  32  and Error term is  6.794511958837992
iteration is  33  and Error term is  6.132718692188198
iteration is  34  and Error term is  5.534693441936662
iteration is  35  and Error term is  4.9947339062372205
iteration is  36  and Error term is  4.505422658659432
iteration is  37  and Error term is  4.06000253828608
iteration is  38  and Error term is  3.65402859859463
iteration is  39  and Error term is  3.2835645424363906
iteration is  40  and Error term is  2.9457957875029663
iteration is  41  and Error term is  2.6391425639562973
iteration is  42  and Error term is  2.362528403226571
iteration is  43  and Error term is  2.1150719983446926
iteration is  44  and Error term is  1.8953629231229878
iteration is  45  and Error term is  1.7013876165972117
iteration is  46  and Error term is  1.5302798225278205
iteration is  47  and Error term is  1.377525028198638
iteration is  48  and Error term is  1.2418433820183594
iteration is  49  and Error term is  1.1198974079407147
iteration is  50  and Error term is  1.0111401033937606
iteration is  51  and Error term is  0.9129708372279185
iteration is  52  and Error term is  0.8242263351746695
iteration is  53  and Error term is  0.7447237610883253
iteration is  54  and Error term is  0.6731735685563812
iteration is  55  and Error term is  0.6082173511591665
iteration is  56  and Error term is  0.5492542799542753
iteration is  57  and Error term is  0.4963525424422217
iteration is  58  and Error term is  0.4488198141753264
iteration is  59  and Error term is  0.40603876241103354
iteration is  60  and Error term is  0.36732128893666455
iteration is  61  and Error term is  0.3321619028454681
iteration is  62  and Error term is  0.3001888451358104
iteration is  63  and Error term is  0.27115177263984974
iteration is  64  and Error term is  0.24489412954905201
iteration is  65  and Error term is  0.22129735782467094
iteration is  66  and Error term is  0.2000705762055484
iteration is  67  and Error term is  0.1808987845640201
iteration is  68  and Error term is  0.16356573590106066
iteration is  69  and Error term is  0.1478796434294466
iteration is  70  and Error term is  0.1336763921712208
iteration is  71  and Error term is  0.12082977710311496
iteration is  72  and Error term is  0.1092040051977595
iteration is  73  and Error term is  0.09866658500981441
iteration is  74  and Error term is  0.08914857698951069
iteration is  75  and Error term is  0.08059848837908562
iteration is  76  and Error term is  0.07293803549913551
iteration is  77  and Error term is  0.06607450510446104
iteration is  78  and Error term is  0.05991060272790561
iteration is  79  and Error term is  0.05434088163332077
iteration is  80  and Error term is  0.049285381961272044
iteration is  81  and Error term is  0.04469588704250972
iteration is  82  and Error term is  0.040530341481279104
iteration is  83  and Error term is  0.03675107846293224
iteration is  84  and Error term is  0.03332339947705845
iteration is  85  and Error term is  0.030216255751221714
iteration is  86  and Error term is  0.027396002254669885
iteration is  87  and Error term is  0.024831498336977006
iteration is  88  and Error term is  0.022501702583424453
iteration is  89  and Error term is  0.020389382323795648
iteration is  90  and Error term is  0.018476129896170496
iteration is  91  and Error term is  0.016743251603127472
iteration is  92  and Error term is  0.015174064962778165
iteration is  93  and Error term is  0.013753247062329934
iteration is  94  and Error term is  0.012466799550028987
iteration is  95  and Error term is  0.01130118470772464
iteration is  96  and Error term is  0.010243195482809192
iteration is  97  and Error term is  0.00928200669248555
iteration is  98  and Error term is  0.008408891896174978
iteration is  99  and Error term is  0.007616004606704308
iteration is  100  and Error term is  0.006896900864576436
iteration is  101  and Error term is  0.006245512186688495
iteration is  102  and Error term is  0.005655635183997234
iteration is  103  and Error term is  0.005121497255545205
iteration is  104  and Error term is  0.004637886238136942
iteration is  105  and Error term is  0.004200091712822389
iteration is  106  and Error term is  0.003803831066793984
iteration is  107  and Error term is  0.0034452253408612378
iteration is  108  and Error term is  0.003120752941328071
iteration is  109  and Error term is  0.002827226115066814
iteration is  110  and Error term is  0.002561757013124554
iteration is  111  and Error term is  0.0023217285316735575
iteration is  112  and Error term is  0.0021047673463345812
iteration is  113  and Error term is  0.001908692728648706
iteration is  114  and Error term is  0.0017315540796133612
iteration is  115  and Error term is  0.0015714936318885402
iteration is  116  and Error term is  0.0014267682473917578
iteration is  117  and Error term is  0.0012957807542168106
iteration is  118  and Error term is  0.0011770692393954726
iteration is  119  and Error term is  0.0010693189929280966
iteration is  120  and Error term is  0.0009715290313324701
iteration is  121  and Error term is  0.0008827339299621799
iteration is  122  and Error term is  0.0008020556086112714
iteration is  123  and Error term is  0.000728707250615221
iteration is  124  and Error term is  0.0006619866857191594
iteration is  125  and Error term is  0.000601269543460046
iteration is  126  and Error term is  0.0005460011940845293
iteration is  127  and Error term is  0.0004956879737554315
iteration is  128  and Error term is  0.0004498885702892915
iteration is  129  and Error term is  0.00040820625304670505
iteration is  130  and Error term is  0.00037028228734202446
iteration is  131  and Error term is  0.00033579059133247716
iteration is  132  and Error term is  0.00030443351172473177
iteration is  133  and Error term is  0.00027593852012186565
iteration is  134  and Error term is  0.0002500556170650333
iteration is  135  and Error term is  0.0002265552552587508
iteration is  136  and Error term is  0.00020522662925170885
iteration is  137  and Error term is  0.00018587621809944025
iteration is  138  and Error term is  0.00016832650232290376
iteration is  139  and Error term is  0.00015241480093973657
iteration is  140  and Error term is  0.00013799219540868636
iteration is  141  and Error term is  0.00012492252138931353
iteration is  142  and Error term is  0.00011308141661097233
iteration is  143  and Error term is  0.00010235542047866982
iteration is  144  and Error term is  9.264112207212626e-05
iteration is  145  and Error term is  8.384435652972197e-05
iteration is  146  and Error term is  7.587944973110667e-05
iteration is  147  and Error term is  6.866851015985712e-05
iteration is  148  and Error term is  6.214076891246526e-05
iteration is  149  and Error term is  5.623196484022768e-05
iteration is  150  and Error term is  5.0883776362478124e-05
iteration is  151  and Error term is  4.604329509733521e-05
iteration is  152  and Error term is  4.166254236062395e-05
iteration is  153  and Error term is  3.7698023761286684e-05
iteration is  154  and Error term is  3.4110321235545106e-05
iteration is  155  and Error term is  3.086372000750748e-05
iteration is  156  and Error term is  2.792586752857811e-05
iteration is  157  and Error term is  2.5267461755080753e-05
iteration is  158  and Error term is  2.2861967968253335e-05
iteration is  159  and Error term is  2.068536049624401e-05
iteration is  160  and Error term is  1.8715887791416153e-05
iteration is  161  and Error term is  1.6933859175228303e-05
iteration is  162  and Error term is  1.53214508089017e-05
iteration is  163  and Error term is  1.3862530277247316e-05
iteration is  164  and Error term is  1.2542496378061458e-05
iteration is  165  and Error term is  1.1348134790684757e-05
iteration is  166  and Error term is  1.0267486845718118e-05
iteration is  167  and Error term is  9.289730302979291e-06
iteration is  168  and Error term is  8.40507219015962e-06
iteration is  169  and Error term is  7.604650502664606e-06
iteration is  170  and Error term is  6.880446409902088e-06
iteration is  171  and Error term is  6.225204018720825e-06
iteration is  172  and Error term is  5.63235762691311e-06
iteration is  173  and Error term is  5.095966469208384e-06
iteration is  174  and Error term is  4.610655346326543e-06
iteration is  175  and Error term is  4.171560404394463e-06
iteration is  176  and Error term is  3.7742809653693257e-06
iteration is  177  and Error term is  3.414835212564135e-06
iteration is  178  and Error term is  3.0896204569114575e-06
iteration is  179  and Error term is  2.795377039643345e-06
iteration is  180  and Error term is  2.529155604656228e-06
iteration is  181  and Error term is  2.288287552685445e-06
iteration is  182  and Error term is  2.070358665214905e-06
iteration is  183  and Error term is  1.8731842053924212e-06
iteration is  184  and Error term is  1.6947877403880837e-06
iteration is  185  and Error term is  1.5333811015854592e-06
iteration is  186  and Error term is  1.387346127270166e-06
iteration is  187  and Error term is  1.2552190419629498e-06
iteration is  188  and Error term is  1.1356752832116564e-06
iteration is  189  and Error term is  1.0275164580695718e-06
iteration is  190  and Error term is  9.296583459057608e-07
iteration is  191  and Error term is  8.411199399401749e-07
iteration is  192  and Error term is  7.610136834763229e-07
iteration is  193  and Error term is  6.885365211773941e-07
iteration is  194  and Error term is  6.229618536460992e-07
iteration is  195  and Error term is  5.636323840452894e-07
iteration is  196  and Error term is  5.099532483308441e-07
iteration is  197  and Error term is  4.6138644844401135e-07
iteration is  198  and Error term is  4.17444958410823e-07
iteration is  199  and Error term is  3.7768842935398683e-07
iteration is  200  and Error term is  3.417181512709929e-07
iteration is  201  and Error term is  3.0917369330924925e-07
iteration is  202  and Error term is  2.797285931769758e-07
iteration is  203  and Error term is  2.5308782662290734e-07
iteration is  204  and Error term is  2.2898431359353193e-07
iteration is  205  and Error term is  2.071762471217025e-07
iteration is  206  and Error term is  1.8744519592738163e-07
iteration is  207  and Error term is  1.695933627893029e-07
iteration is  208  and Error term is  1.5344157078332136e-07
iteration is  209  and Error term is  1.3882814758306414e-07
iteration is  210  and Error term is  1.2560641144860583e-07
iteration is  211  and Error term is  1.1364391024864586e-07
iteration is  212  and Error term is  1.0282068756664399e-07
iteration is  213  and Error term is  9.302821295002828e-08
iteration is  214  and Error term is  8.416839465686991e-08
iteration is  215  and Error term is  7.615237185204938e-08
iteration is  216  and Error term is  6.889973610741334e-08
iteration is  217  and Error term is  6.233789682678046e-08
iteration is  218  and Error term is  5.640092645543799e-08
iteration is  219  and Error term is  5.1029487576229526e-08
iteration is  220  and Error term is  4.6169517193644627e-08
iteration is  221  and Error term is  4.177236071458647e-08
iteration is  222  and Error term is  3.7794061767050025e-08
iteration is  223  and Error term is  3.4194701521793885e-08
iteration is  224  and Error term is  3.093802910986831e-08
iteration is  225  and Error term is  2.7991513597837628e-08
iteration is  226  and Error term is  2.5325667322783733e-08
iteration is  227  and Error term is  2.291373747184729e-08
iteration is  228  and Error term is  2.0731451578392997e-08
iteration is  229  and Error term is  1.8757053463294803e-08
iteration is  230  and Error term is  1.697062794247227e-08
iteration is  231  and Error term is  1.5354362073591056e-08
iteration is  232  and Error term is  1.389204826372071e-08
iteration is  233  and Error term is  1.2569027100806917e-08
iteration is  234  and Error term is  1.1371987969748516e-08
iteration is  235  and Error term is  1.0288952116634768e-08
iteration is  236  and Error term is  9.309039605482992e-09
Utility 1: DONE
Utility 2: DONE
```

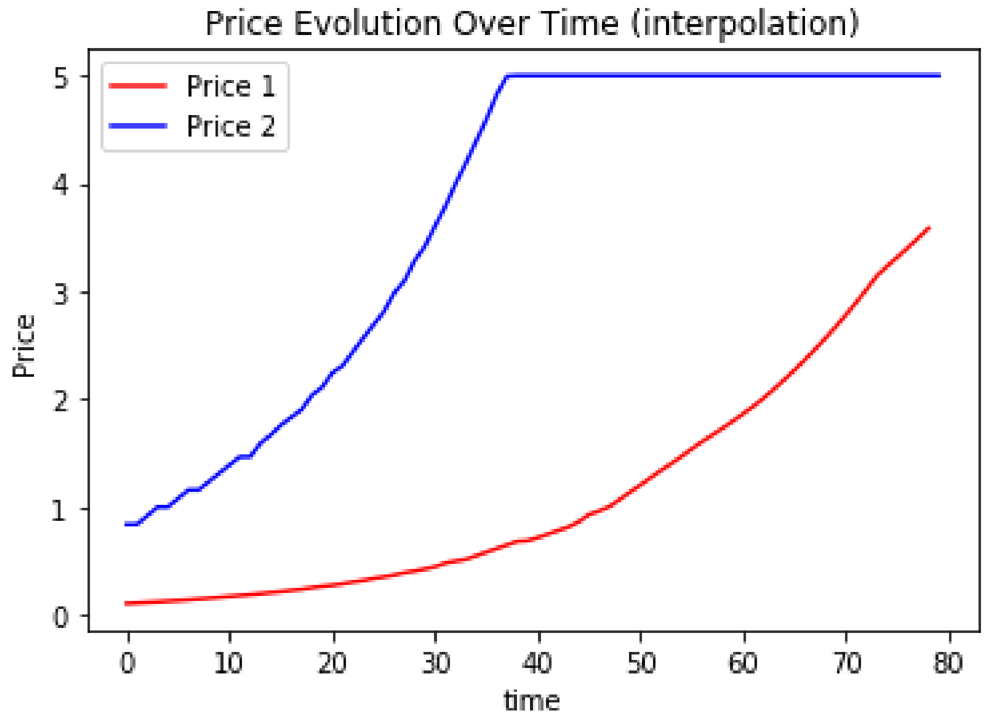
[14]:
```
### Find the optimal Transition matrices
T_opt1_v2 = np.zeros((N2, N2))
for ik in range(N2):
    diff = S2[ik] - A2[int(IndexAction1_v2[ik])]
    dist = abs(S2 - diff) # How far from the differenre is each element of S2
    index1 = np.where(dist == sorted(dist)[0])[0][0] # Find the index of the␣
 ↪closest
    index2 = np.where(dist == sorted(dist)[1])[0][0] # Find the index of the␣
 ↪second closest
    weight1 = sorted(dist)[0] / np.sum(sorted(dist)[0:2])
    weight2 = sorted(dist)[1] / np.sum(sorted(dist)[0:2])
    weight1 + weight2 == 1
    T_opt1_v2[ik, index1] = weight1
    T_opt1_v2[ik, index2] = weight2

T_opt1_v2 = csr_matrix(T_opt1_v2)

T_opt2_v2 = np.zeros((N2, N2))
for ik in range(N2):
    diff = S2[ik] - A2[int(IndexAction2_v2[ik])]
    dist = abs(S2 - diff) # How far from the differenre is each element of S2
    index1 = np.where(dist == sorted(dist)[0])[0][0] # Find the index of the␣
 ↪closest
    index2 = np.where(dist == sorted(dist)[1])[0][0] # Find the index of the␣
 ↪second closest
    weight1 = sorted(dist)[0] / np.sum(sorted(dist)[0:2])
    weight2 = sorted(dist)[1] / np.sum(sorted(dist)[0:2])
    weight1 + weight2 == 1
    T_opt2_v2[ik, index1] = weight1
    T_opt2_v2[ik, index2] = weight2

T_opt2_v2 = csr_matrix(T_opt2_v2)
```

[15]:
```
### Simulate

St1_v2 = 1000*np.ones(80)
C1_v2 = np.zeros(80)
for i in range(80-1):
    #init_index1 = np.where(S2==St1_v2[i])[0][0]
    init = St1_v2[i]
    # Find closest states
    dist = abs(S2 - init)
    tokeep = sorted(dist)[0:2]
    index1 = np.where(dist == tokeep[0])[0][0]
    index2 = np.where(dist == tokeep[1])[0][0]
    weight1 = 1 - tokeep[0] / np.sum(tokeep[0:2])
    weight2 = 1 - tokeep[1] / np.sum(tokeep[0:2])
    weight1 + weight2 == 1
    C1_v2[i] = A2[int(IndexAction1_v2[index1])]*weight1 +␣
 ↪A2[int(IndexAction1_v2[index2])]*weight2
    St1_v2[i+1] = St1_v2[i] - C1_v2[i]
p1_v2 = p1_f(C1_v2)

# Simulate
St2_v2 = 1000*np.ones(80)
C2_v2 = np.zeros(80)
for i in range(80-1):
    #init_index1 = np.where(S2==St1_v2[i])[0][0]
    init = St2_v2[i]
    # Find closest states
    dist = abs(S2 - init)
    tokeep = sorted(dist)[0:2]
    index1 = np.where(dist == tokeep[0])[0][0]
    index2 = np.where(dist == tokeep[1])[0][0]
    weight1 = 1 - tokeep[0] / np.sum(tokeep[0:2])
    weight2 = 1 - tokeep[1] / np.sum(tokeep[0:2])
    weight1 + weight2 == 1
    C2_v2[i] = A2[int(IndexAction2_v2[index1])]*weight1 +␣
 ↪A2[int(IndexAction2_v2[index2])]*weight2
    St2_v2[i+1] = St2_v2[i] - C2_v2[i]

p2_v2 = p2_f(C2_v2)
```

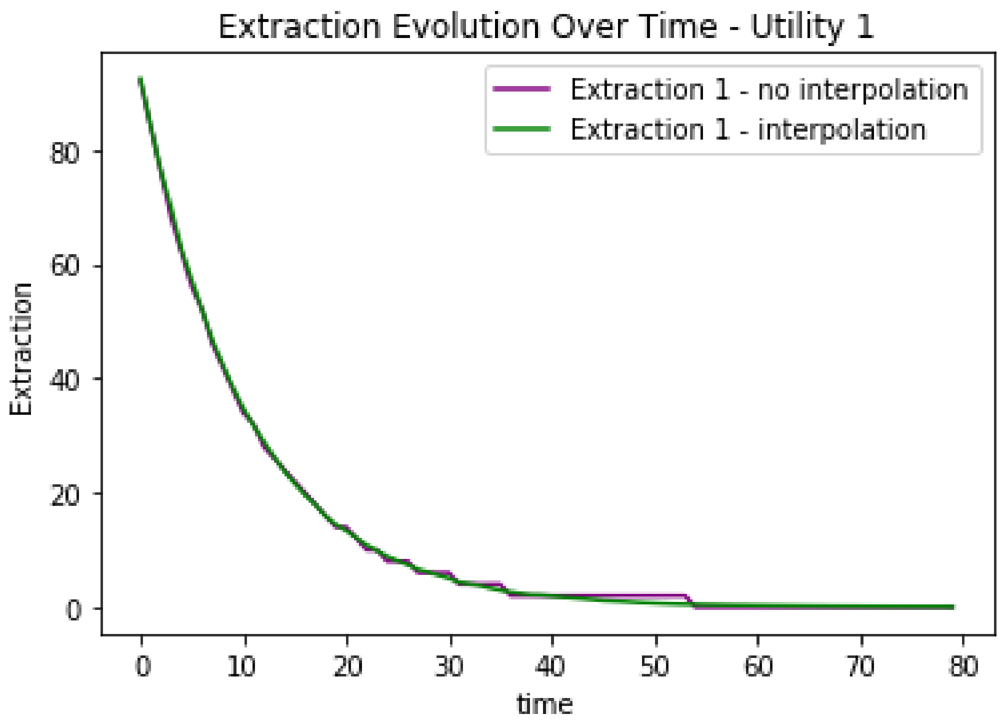
```
/Applications/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:47:
RuntimeWarning: divide by zero encountered in true_divide
```

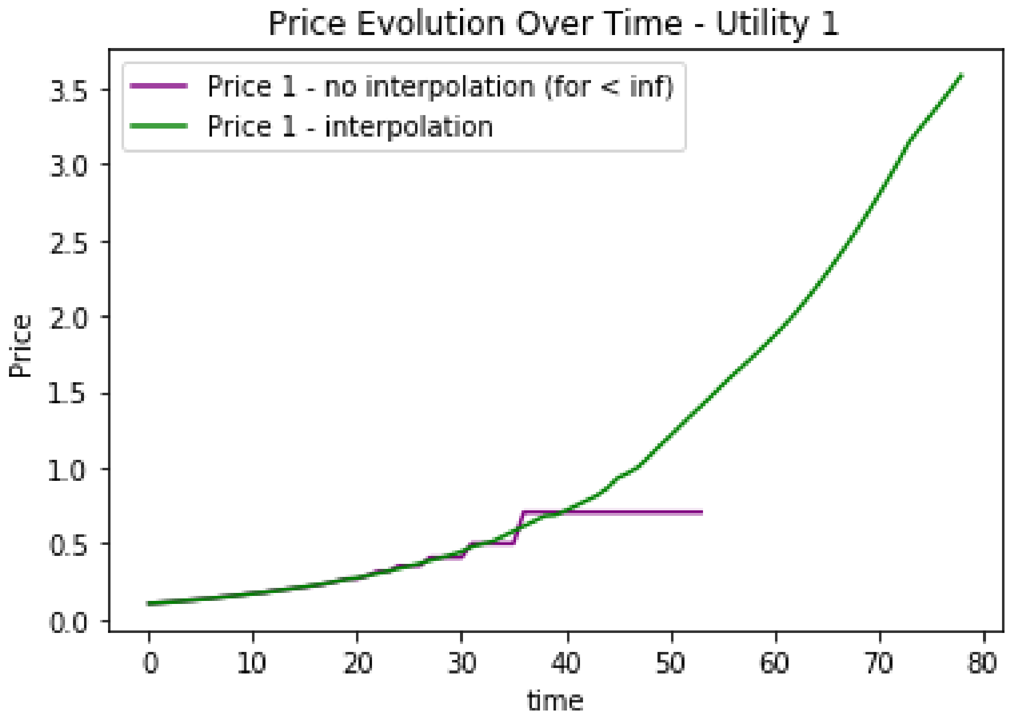
[16]:
```
plt.plot(range(0,80), C1_v2, color="red", label="Extraction 1")
plt.plot(range(0,80), C2_v2, color="blue", label="Extraction 2")
plt.title("Extraction Evolution Over Time (interpolation)")
plt.xlabel("time")
plt.ylabel("Extraction")
plt.legend()
plt.show()



# For printing purposes
plt.plot(range(0,80), p1_v2, color="red", label="Price 1")
plt.plot(range(0,80), p2_v2, color="blue", label="Price 2")
plt.title("Price Evolution Over Time (interpolation)")
plt.xlabel("time")
plt.ylabel("Price")
plt.legend()
plt.show()
```

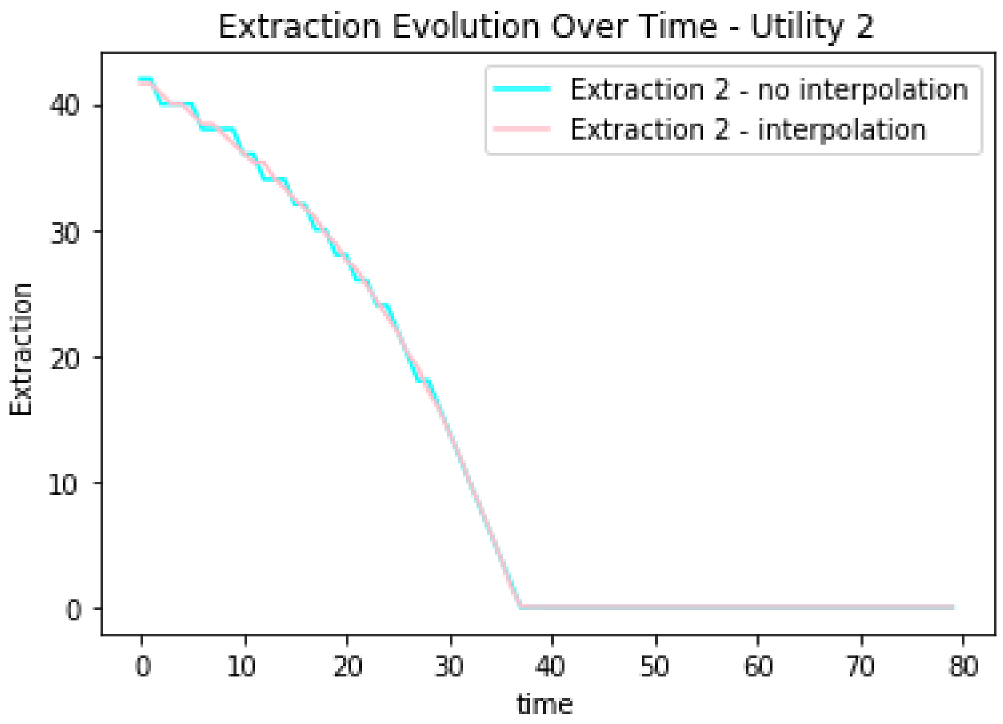

Extraction Evolution Over Time (interpolation)
Extraction 1
Extraction 2
Extraction
0
20
40
60
80
0
10
20
30
40
50
60
70
80
time


Price Evolution Over Time (interpolation)
Price 1
Price 2
Price
0
1
2
3
4
5
0
10
20
30
40
50
60
70
80
time

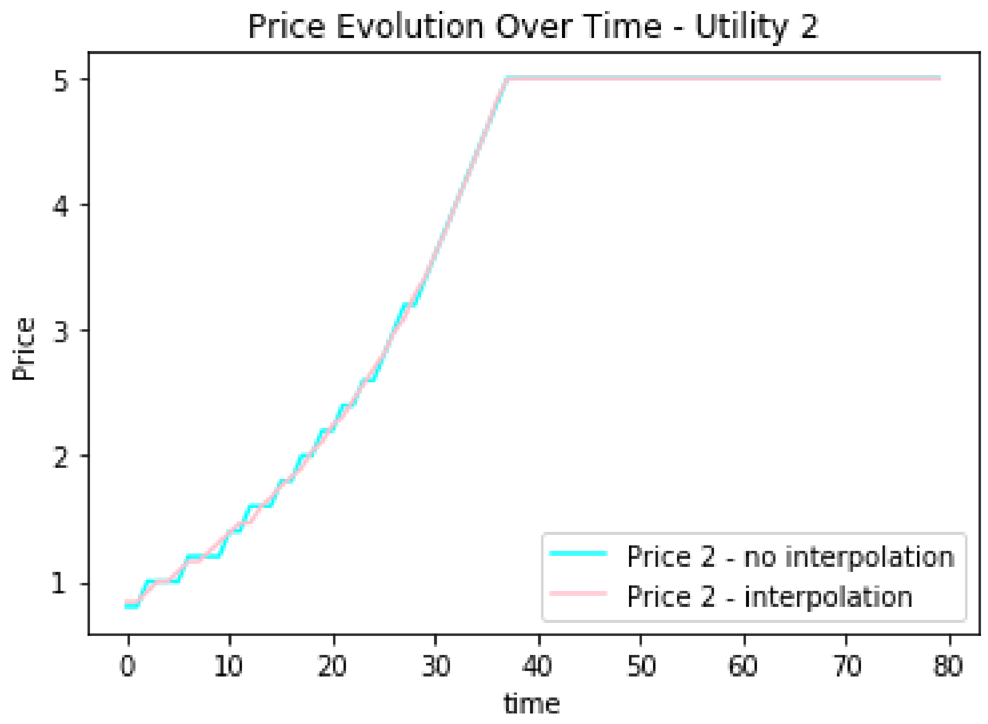
```
[17]: # For Free: a comparison of the interpolation & non-interpolation outcomes
plt.plot(range(0,80), C1_v1, color="purple", label="Extraction 1 - no␣
 ↪interpolation")
plt.plot(range(0,80), C1_v2, color="green", label="Extraction 1 -␣
 ↪interpolation")
plt.title("Extraction Evolution Over Time - Utility 1")
plt.xlabel("time")
plt.ylabel("Extraction")
plt.legend()
plt.show()

plt.plot(range(0,80), p1_v1, color="purple", label="Price 1 - no interpolation␣
 ↪(for < inf)")
plt.plot(range(0,80), p1_v2, color="green", label="Price 1 - interpolation")
plt.title("Price Evolution Over Time - Utility 1")
plt.xlabel("time")
plt.ylabel("Price")
plt.legend()
plt.show()


plt.plot(range(0,80), C2_v1, color="cyan", label="Extraction 2 - no␣
 ↪interpolation")
plt.plot(range(0,80), C2_v2, color="pink", label="Extraction 2 - interpolation")
plt.title("Extraction Evolution Over Time - Utility 2")
plt.xlabel("time")
plt.ylabel("Extraction")
plt.legend()
plt.show()

plt.plot(range(0,80), p2_v1, color="cyan", label="Price 2 - no interpolation")
plt.plot(range(0,80), p2_v2, color="pink", label="Price 2 - interpolation")
plt.title("Price Evolution Over Time - Utility 2")
plt.xlabel("time")
plt.ylabel("Price")
plt.legend()
plt.show()
```


Extraction Evolution Over Time - Utility 1
Extraction 1 - no interpolation
Extraction 1 - interpolation
Extraction
80
60
40
20
0
0
10
20
30
40
50
60
70
80
time


Price Evolution Over Time - Utility 1
Price 1 - no interpolation (for < inf)
Price 1 - interpolation
Price
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50
60
70
80
time



Extraction Evolution Over Time - Utility 2
Extraction 2 - no interpolation
Extraction 2 - interpolation
Extraction
0
10
20
30
40
time
0
10
20
30
40
50
60
70
80



Price Evolution Over Time - Utility 2
Price 2 - no interpolation
Price 2 - interpolation
Price
1
2
3
4
5
time
0
10
20
30
40
50
60
70
80

A couple remarks about the differences between problem 1 and problem 2: - Interestingly, with utility 1, notice that the price goes to ∞ in problem 1, but not in problem 2. The intuition is that with this utility, $u_1'(0) = \infty$, so if the agent is not constrained by the grid, he will make sure to always extract a strictly positive amount every period. When $t \to \infty$ we should converge to 0 stock, but always strictly positive. The grid interpolation approximates a continuous problem. - For utility 2, we do not observe the same phenomenon because u_2' is bounded about by 5.