Problem Set #1

Econ Theory, Jason Debacker
Alex Weinberg

Problem 1

Consider the problem of the owner of an oil field. The owner has B barrels of oil. She can sell these barrels at price pt at time t. Her objective is to maximize the discounted present value of sales of oil - well assume there are no extraction costs. The owner discounts the future at a rate given by 1/1+r (where r is the real interest rate and assumed to be constant). Answer the following:

1. What are the state variables?
2. What are the control variables?
3. What does the transition equation look like?
4. Write down the sequence problem of the owner. Write down the Bellman equation.
5. What does the owners Euler equation like?
6. What would the solution of the problem look like if pt+1 = pt for all t? What would the solution look like if pt+1 ¿ (1 + r)pt for all t? What is the condition on the path of prices necessary for an interior solution (where the owner will extract some, but not all, of the oil)?

Solution 1.

1. The state variable is B. The prices r,p_t are given exogenously.
2. Control variables are B' (oil to save) and b (oil to sell today)
3. Transition equation is B' = B - b
4. Sequence problem is:
$$\max \sum_{i=0}^{\infty} \frac{b_t p_t}{(1+r)^t}$$
Bellman equation is:
$$V(B) = p_t b_t + \quad \frac{1}{1+r} V(B')$$
5. Euler equation is:
$$p_t = \frac{1}{1+r} p_{t+1}$$
6.
   - If $p_{t+1} = p_t = p \quad \forall t$ then the owner will sell all oil today.
   - If $p_{t+1} > (1+r)p_t \quad \forall t$ then she always saves all B til tomorrow.
   - $p_t = \frac{1}{1+r} p_{t+1}$ is the necessary condition for the owner to be indifferent between extracting today and extracting tomorrow.

Problem 2

The Neoclassical Growth Model is a workhorse model in macroeconomics. The problem for the social planner is to maximize the discounted expected utility for agents in the economy: maxE tu(c) (1) c0 t t=0 t=0 1 The resource constraint is given as: yt = ct + it (2) The law of motion for the capital stock is: kt+1 =(1)kt +it (3) Output is determined by the aggregate production function: yt = ztkt (4) Assume that zt is stochastic. In particular, it is an i.i.d. process distributed as ln(z) N(0,z).

Solution 2.

1. State variable is k_t, z_t
2. Control variables are c_t, i_t can reduce to just c_t
3. $V(k_t, z_t) = u(c_t) + \beta E_t[V(k_{t+1}, z_{t+1}) \mid z_t]$
   s.t.

$$y_t = c_t + i_t \tag{1}$$
$$k_{t+1} = (1-\delta)k_t + i_t \tag{2}$$
$$y_t = z_t k_t^{\alpha} \tag{3}$$

```
# Import packages
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt
from scipy.optimize import fminbound
from scipy import interpolate
from quantecon.markov.approximation import rouwenhorst

# Quesiton 2
## PARAMETERS
gamma = 0.5
beta = 0.96
delta = 0.05
alpha = 0.4
sigz = 0.2
muz = 0

# Discretize captial
kmin = 10
kmax = 13
nk = 30

kgrid = np.linspace(kmin,kmax,nk)

# Discretize risk
```

```
nz = 30
zdist = rouwenhorst(nz, muz, sigz, rho=0)
zgrid = np.exp(zdist.state_values)
pi = zdist.P

# Options
tol = 1e-4
maxiter = 1000


'''
 ---------------------------------------------------------------------------------
Create grid of current utility values
-----------------------------------------------------------------------
C       = matrix, current consumption (c=z_tk_t^a - k_t+1 + (1-delta)k_t)
U       = matrix, current period utility value for all possible
          choices of w and w' (rows are w, columns w')
-----------------------------------------------------------------------
'''

C = np.zeros((nk, nk, nz))
for i in range(nk): # loop over k_t
    for j in range(nk): # loop over k_t+1
        for q in range(nz): #loop over z_t
            C[i, j, q] = zgrid[q]* kgrid[i]**alpha + (1 - delta)*kgrid[i] -
                kgrid[j]
# replace 0 and negative consumption with a tiny value
# This is a way to impose non-negativity on cons
C[C<=0] = 1e-15
if gamma == 1:
    U = np.log(C)
else:
    U = (C ** (1 - gamma)) / (1 - gamma)
U[C<0] = -9999999


def production(k,z=1):
    y = z * (k ** alpha)
    return y

def capital_transition(k,sav):
    knew = (1 - delta) * k + sav
    return knew

def expected_value(Vlast,k,iz,sav):
    '''
    V = value func
```

```
    k = current capital
    iz = index of current shock
    Takes in value function and current state and spits out
    expected_value for each savings decision
    '''

    EV = 0
    for ii, z_prime in enumerate(zgrid):
        V_func = interpolate.interp1d(kgrid, Vlast[:,ii], kind='cubic',
            fill_value='extrapolate')

        k_tomo = capital_transition(k,sav)

        EV += pi[iz, ii] * V_func(k_tomo)
    return EV
################
#VFI
################
'''
------------------------------------------------------------------------
Value Function Iteration
------------------------------------------------------------------------
VFtol     = scalar, tolerance required for value function to converge
VFdist    = scalar, distance between last two value functions
VFmaxiter = integer, maximum number of iterations for value function
V         = vector, the value functions at each iteration
Vmat      = matrix, the value for each possible combination of w and w'
Vstore    = matrix, stores V at each iteration
VFiter    = integer, current iteration number
TV        = vector, the value function after applying the Bellman operator
PF        = vector, indicies of choices of w' for all w
VF        = vector, the "true" value function
------------------------------------------------------------------------
'''
VFtol = 1e-4
VFdist = 7.0
VFmaxiter = 500
V = np.zeros((nk, nz)) # initial guess at value function
Vmat = np.zeros((nk, nk, nz)) # initialize Vmat matrix
Vstore = np.zeros((nk, nz, VFmaxiter)) #initialize Vstore array
VFiter = 1
while VFdist > VFtol and VFiter < VFmaxiter:
    print('Iteration', VFiter, 'Distance,', VFdist)
    for i in range(nk): # loop over k_t
        for j in range(nk): # loop over k_t+1
            for q in range(nz): #loop over z_t
                EV = 0
```

```
            for qq in range(nz):
                EV += pi[q, qq]*V[j, qq]
            Vmat[i, j, q] = U[i, j, q] + beta * EV

    Vstore[:,:, VFiter] = V.reshape(nk, nz,) # store value function at
        each iteration for graphing later
    TV = Vmat.max(1) # apply max operator over k_t+1
    PF = np.argmax(Vmat, axis=1)
    VFdist = (np.absolute(V - TV)).max() # check distance
    V = TV
    VFiter += 1



if VFiter < VFmaxiter:
    print('Value function converged after this many iterations:', VFiter)
else:
    print('Value function did not converge')


VF = V # solution to the functional equation

# Plot value function
plt.figure()
fig, ax = plt.subplots()
ax.plot(kgrid[1:], VF[1:, 0], label='$z$ = ' + str(kgrid[0]))
ax.plot(kgrid[1:], VF[1:, 5], label='$z$ = ' + str(kgrid[5]))
ax.plot(kgrid[1:], VF[1:, 15], label='$z$ = ' + str(kgrid[15]))
ax.plot(kgrid[1:], VF[1:, 19], label='$z$ = ' + str(kgrid[19]))
# Now add the legend with some customizations.
legend = ax.legend(loc='lower right', shadow=False)
# Set the fontsize
for label in legend.get_texts():
    label.set_fontsize('large')
for label in legend.get_lines():
    label.set_linewidth(1.5) # the legend line width
plt.xlabel('Size of Capital')
plt.ylabel('Value Function')
plt.title('Value Function')
plt.savfig('1.png')
plt.show()

#Plot optimal consumption rule as a function of capital
optK = kgrid[PF]
optC = kgrid * kgrid ** (alpha) + (1 - delta) * kgrid - optK
plt.figure()
fig, ax = plt.subplots()
```

```
ax.plot(kgrid[:], optC[:][18], label='Consumption')
# Now add the legend with some customizations.
#legend = ax.legend(loc='upper left', shadow=False)
# Set the fontsize
for label in legend.get_texts():
    label.set_fontsize('large')
for label in legend.get_lines():
    label.set_linewidth(1.5) # the legend line width
plt.xlabel('Size of Capital')
plt.ylabel('Optimal Consumption')
plt.title('Policy Function, consumption - growth model')
plt.savfig('2.png')
plt.show()

#Plot optimal capital in period t + 1 rule as a function of cake size
optK = kgrid[PF]
plt.figure()
fig, ax = plt.subplots()
ax.plot(kgrid[:], optK[:][18], label='Capital in period t+1')
# Now add the legend with some customizations.
#legend = ax.legend(loc='upper left', shadow=False)
# Set the fontsize
for label in legend.get_texts():
    label.set_fontsize('large')
for label in legend.get_lines():
    label.set_linewidth(1.5) # the legend line width
plt.xlabel('Size of Capital in period t')
plt.ylabel('Optimal Capital in period t+1')
plt.title('Policy Function, capital next period - growth model')
plt.savfig('3.png')
plt.show()
```

Solution 3. Change rho = 0.8 in above.

$$V(w) = \max\{V^U(w), V^J(w)\}$$

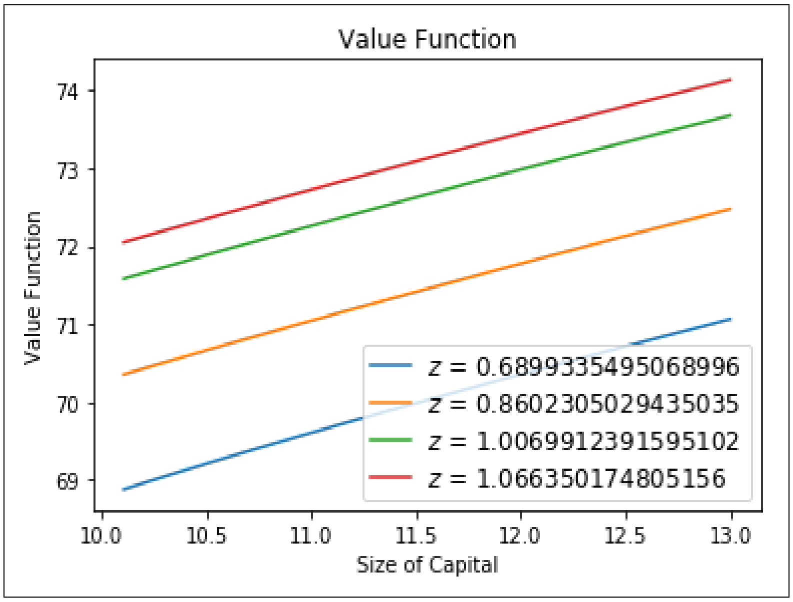
where:

$$V^U(w) = b + \beta EV(w)$$

and

$$V^J(w) = E_0 \sum_{t=0}^{\infty} \beta^t w = \frac{w}{1-\beta}$$

Figure 1: Great example figure



Solution 4. See LaborDP.ipyb for answers and code.

Figure 2: Great example figure

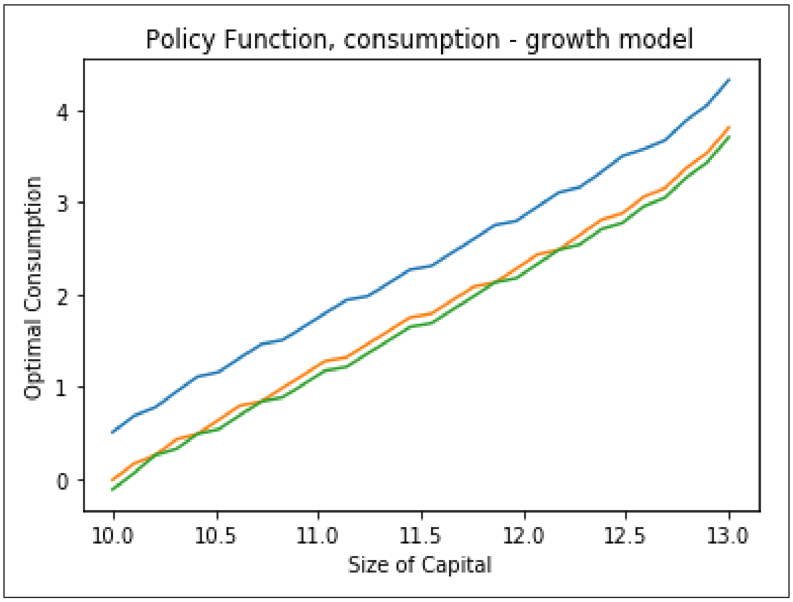


Figure 3: Great example figure

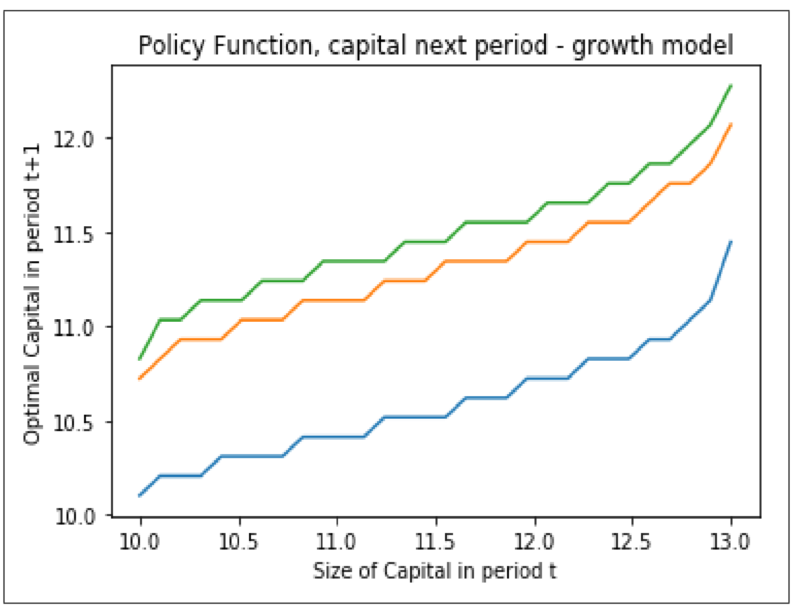


Figure 4: Great example figure

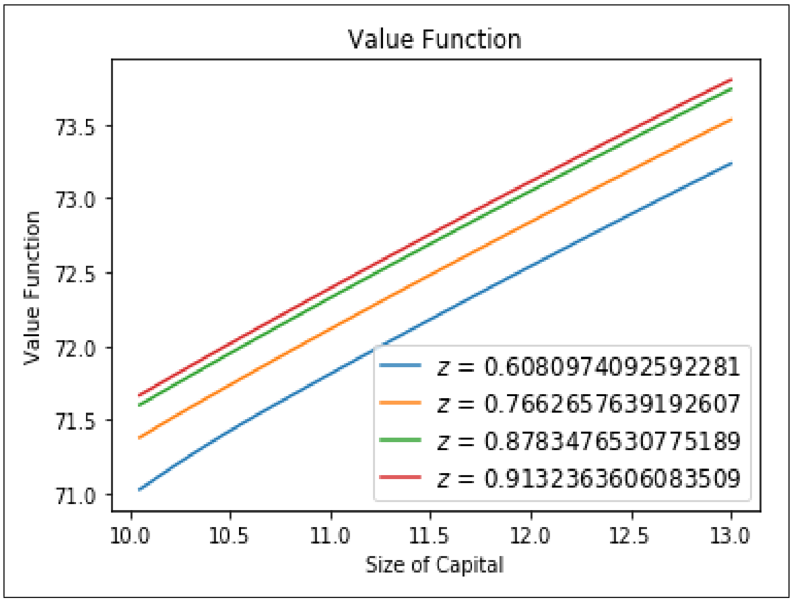


Figure 5: Great example figure

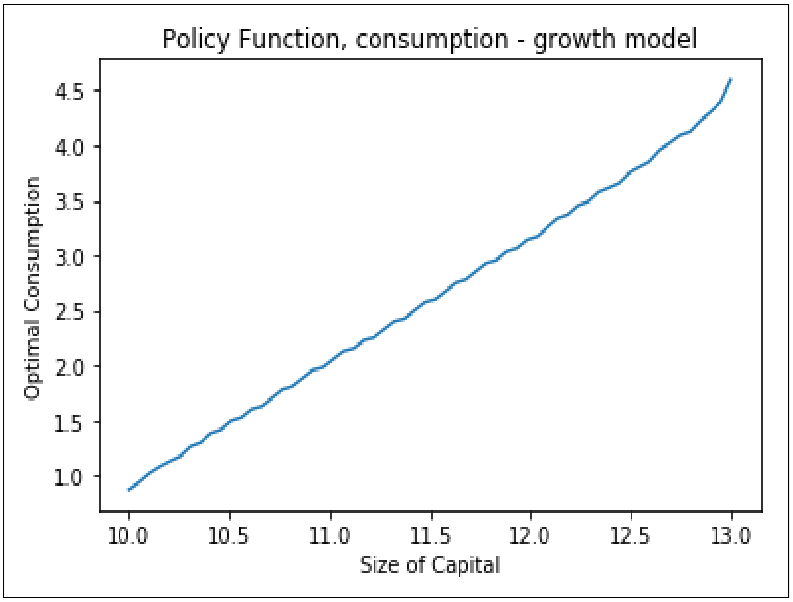


Figure 6: Great example figure