Numerical Methods in Economics: Final Project

R Yang

1 Problem

We want to find a numerical solution to a life-cycle problem where people save, face stochastic wages, have mean wages that follow an exogenous pattern, have elastic labor supply, cannot borrow, and have a bequest motive.

This takes the form of a finite period dynamic programming problem with two continuous controls in each period, savings (k_{t+1}) and labor (l). The state is the amount of capital that the agent comes into each period with, k_t, and furthermore, depending on the nature of the stochastic wages, there might be more states.

2 Plan

We hope to solve this problem through approximation of the value function at each time period, and backwards induction. We will approximate with Chebyshev polynomials at Chebyshev nodes. The following is our procedure:

1. Compute the value in period T, v_i^t for each x_i. This amounts to solving $\max_{k_{T+1},l_T} u(c_T, l_T) + \beta \cdot u_T(k_{T+1})$.
2. Using the (v_i, x_i) points from step 1, approximate the value function at time t with a Chebyshev polynomial to get V^T.
3. Compute the value in period T-1, i.e. solve $\max_{k_T,l_{T-1}} u(c_{T-1}, l_{T-1}) + \beta V^T(k_T)$, v_i^t for each x_i.
4. Using the value from step 3, approximate this value function at time T-1.
5. Repeat steps 3 and 4 until we have an approximation for V^1.

3 Code

We expose one main function in stochdy.py: stochdy.execute(deg, pts, opts, preserveShape). Given a polynomial degree (deg), number of approximation points (pts), and a dictionary of options (opts), res=stochdy.execute returns a list of arrays, where res[t][i] is an array of coefficients describing a chebyshev polynomial which approximates the value function at time t and in wage state i.

The following describes the entries to specify a model within the opts dict to be executed by the stochdy module:

Model Features

- opts['utility']: A function which takes two arrays, where the first array is an nControl-length array of the value of the controls (e.g. k_{t+1} and l) and the second array is a 2-length array of [k0, wage], and returns the period's utility. ([Float], [Float] → Float)
- opts['bequest']: A function which takes the leftover capital from the final period and returns the value of that capital. (Float → Float)
- opts['statewage']: A function which takes the current state and the wage and returns the wage for the period. this allows us to handle many different kinds of variation in wages. (Int, Float → Float)
- opts['T']: Number of time periods (Int)

- opts['wages']: An array of wages over time. this array must be of T length. ([Float])
- opts['beta']: Discounting on next period. (Float)
- opts['trans']: A nStates by nStates state-transition matrix for the wage-state process. ([[Float]])
- opts['states']: An array of [0...nStates]. Must be of same length as any dimension of trans. ([Int])

Optimization Options

- opts['bounds']: A function which takes a tuple of [k0, wage] and returns an nControl-length list of tuples, (min, max), for each control. These bounds are used for each control during the maximization step. ((Float, Float) → [(Float, Float)])
- opts['x0']: A function which takes a tuple of [k0, wage] and returns an nControl-length list of the initial guess for each control for the maximization step. ((Float, Float) → [Float])
- opts['init']: Beginning of range (Float)
- opts['end']: End of range (Float)

4 Example

Our example will involve a wage process with three states for wages: high (state 0), mean (state 1), and low (state 2). In the high period, wages are 110% of the mean according to the trend, in the mean period, wages are the trend mean, and in the low period wages are 90% of the mean trend. The transitions between the three states is described by the following matrix:

$$trans = \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.25 & 0.5 & 0.25 \\ 0.1 & 0.4 & 0.5 \end{bmatrix}$$

In order to handle this, we introduce an additional state variable: 0 for high, 1 for mean, and 2 for low. This means that during our maximization step, which is now dependent on the wage state, we will need to compute the value function for our kprime choice given each future state and then compose them together.

The following are the specifications for the entries in the opts dictionary to solve this problem. This example, and others, are in teststochdy.py.

```
def productionDefault(k0, l):
    return k0 ** (1./3.) + (1. - 0.1) * k0

opts['production'] = productionDefault

def utilityDefault(x, y):
    k1, l = x
    k0, w = y

    if l >= 1 or productionDefault(k0,l) - k1 < 0:
        return -1

    return (productionDefault(k0,l) - k1)**0.5 + (1. - l)**0.5

opts['utility'] = utilityDefault
```

This encodes a utility function of the form $u(c_t) = c_t^{\frac{1}{2}} + (1-l)^{\frac{1}{2}}$, with production following $f(k_0) = k_0^{\frac{1}{3}}$ and depreciation of 10%.

```
def bequestValueDefault(x):
    return x**0.5

opts['bequest'] = bequestValueDefault
```

The payoff from holding capital to the end of the 60 periods is the square root of the terminal capital.

```
def statewageDefault(state, wage):
    if state == 0:
        return wage * 0.9
    elif state == 1:
        return wage * 1
    elif state == 2:
        return wage * 1.1

opts['statewage'] = statewageDefault
```

In the low state, wages at 10% lower, which in the high state, wages are 10% higher.

```
opts['T'] = 60
```

Lifetime is 60 one-year periods.

```
wages = np.zeros(opts['T'])
period = int(np.floor(opts['T']/3.))
for i in xrange(period+1):
    wages[i] = i * (5. / period)# growing up to 5
wages[period:2*period] = 5.
for i in xrange(2*period,opts['T']):
    wages[i] = 5. - 2. * (i - 2.*period) / (opts['T'] - 2.*period)

opts['wages'] = wages
```

In the first third of the time period, wages rise from 0 to 5. In the second third of the period, mean wages are constant at 5. In the final third of the period, wages decline from 5 to 3.

```
opts['beta'] = 0.9
```

$\beta = 0.9$

```
mTrans = np.zeros((3,3))
mTrans[0,:] = [0.5,0.4,0.1]
mTrans[1,:] = [0.25,.5,.25]
mTrans[2,:] = [0.1,0.4,0.5]

opts['trans'] = mTrans
```

The transition matrix is as described above.

```
opts['states'] = [0,1,2]
```

Three states, as described above.

```
kbounds = lambda s: (0, opts['production'](s[0],0) + s[1])
lbounds = lambda s: (0,1)

opts['bounds'] = lambda s: (kbounds(s), lbounds(s))
```

We constrain both the choices to be non-negative - there is no borrowing in this model. The maximum labor is 1, while the maximum savings is a function representing production plus wages from maximum labor.

```
opts['x0'] = lambda s: np.array([sum(kbounds(s)) / 2.,sum(lbounds(s)) / 2.])
```

The initial points for the maximization are the midpoints between the bounds for the two choices.

```
opts['init'] = 0.1
opts['end'] = 10
```

We restrict ourselves to the range $[0.1, 10]$