Spencer Lyon

Econ 581 homework 6: Recursive models and Markov processes

Problem 1

We are given the following

- $Y_t = AK_t^\theta$
- $K_{t+1} = I_t$
- $C_t = Y_t - I_t$
- $u(C) = ln(c)$

We can now write the Bellman equation

$$V(K) = \max_K ln(AK^\theta - K') + \beta V(K')$$

The *first order condition* is $u'(c) = \beta * V'(K')$

(if you substitute $K' = I = Y - C$ you get a neagive sign on the V' term. Then set equal to zero and solve for $u'(c)$)

The *envelope condition* is $V'(k) = u'(c)A\theta K^{\theta-1}$

This makes the *Euler Equation* become $u'(c) = \beta u'(c)A\theta K^{\theta-1}$

With log utility $u'(c) = \frac{1}{c}$

The simplified *Euler equation* is
$\frac{C'}{C} = \theta\beta AK'^{\theta-1}$

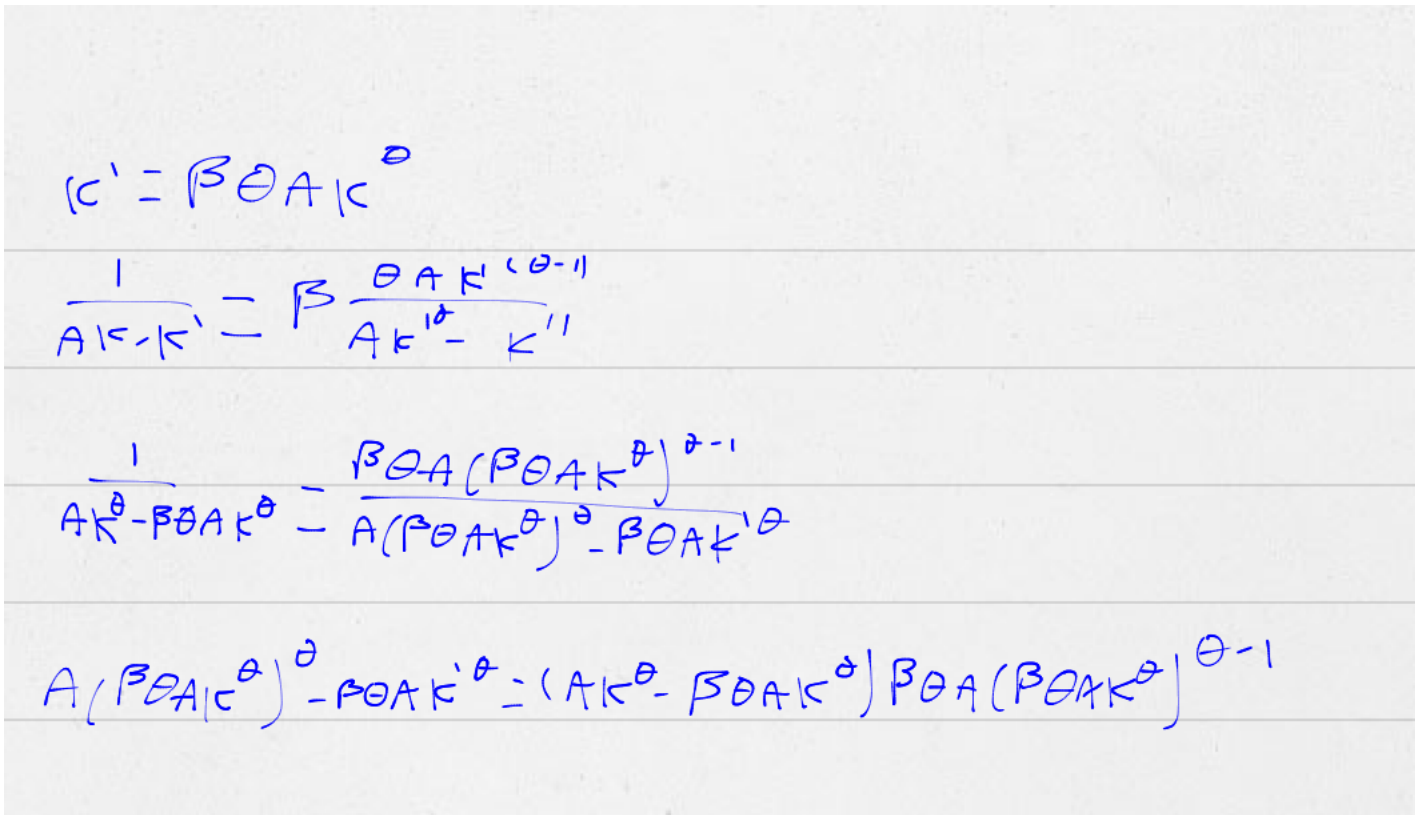
MISSING STEP FOR HOW TO VERIFY POLICY FUNCTION

See the "marcov_p1.png" image below this and just above the "Problem 2" header.

The final form of the value function is the following:
$V(K) = ln(AK^\theta - \beta\theta AK^\theta) + \beta V(\beta\theta AK^\theta)$

In [12]:
```
from IPython.core.display import Image
Image(filename='marcov_p1.png')
```

Out[12]:



$$\beta\theta A k'^{\theta} = A k^{\theta} \beta\theta A (\beta\theta A k^{\theta})^{\theta-1} - (\beta\theta A k^{\theta}) \beta\theta A (\beta\theta A k^{\theta})^{\theta-}$$

$$= -\beta^{\theta}\theta^{\theta} A^{\theta+1} k^{\theta^2} + \beta^{\theta+1}\theta^{\theta+1} A^{\theta+1} k^{\theta^2} + f^{\theta}$$

$$k' = \beta\theta A k^{\theta}$$

Problem 2

In [1]:
```
import numpy as np
import pandas as pd
import pylab as pl
from numpy import asarray
from scipy.optimize import fmin
from pandas.io.data import DataReader
import datetime as dt
from statsmodels.tsa.filters.hp_filter import hpfilter

theta = 0.35
delta = 0.02

start1 = dt.datetime(1947, 1, 1)
end1 = dt.datetime(2012, 4, 1)

output = asarray(DataReader('GDPC1', 'fred',
                            start=start1, end=end1)).squeeze()

differ = (output[1:] - output[:-1]) / output[:-1]
ave_trend = differ.mean()

filtered = hpfilter(np.log(output), lamb=1600)[0]
std_filt = filtered.std(ddof=1)
auto_corr_filt = np.corrcoef(filtered[1:], filtered[:-1], ddof=1)[0, 1]

T = 254
num_sims = 10000
g2 = np.array([0.01, -0.03])
g3 = np.array([0.02, 0.01, -0.03])
marcov2 = np.array([[0.9, 0.1],
                    [0.5, 0.5]]).cumsum(1)

marcov3 = np.array([[0.5, 0.45, 0.05],
                    [0.05, 0.85, 0.10],
                    [0.25, 0.25, 0.5]]).cumsum(1)


def do_mc(periods=T):
    """does mc simulation"""
    s2 = 0
    s3 = 0
    epsilon = np.random.normal(0, 0.02, T)  # generate random shocks

    y2 = np.zeros(periods)
    y3 = np.zeros(periods)

    for t in range(1, periods):
        r_num = np.random.rand()
        y2[t] = g2[s2] + y2[t - 1] + epsilon[t]
        y3[t] = g3[s3] + y3[t - 1] + epsilon[t]
        s2 = np.where(marcov2[s2, :] > r_num)[0][0]
        s3 = np.where(marcov3[s3, :] > r_num)[0][0]

    return np.array([y2, y3])

y2_data = np.zeros((num_sims, T))
y3_data = np.zeros((num_sims, T))
for i in range(num_sims):
    mc_data = do_mc(T)
    y2_data[i, :] = mc_data[0, :]
    y3_data[i, :] = mc_data[1, :]
```

In [2]:
```
def create_moments(data):
    """
    gets the moments he asks for (growth rate, std, autocorr)
    """
    if data.ndim == 2:
        df = pd.DataFrame(data).T
        total_obs = data.shape[0]
        diff = (data[:, 2:] - data[:, 1:-1]) / data[:, 1:-1]
        growth = np.mean(diff, axis=1).mean()
```

```
        filt_data = np.empty(data.shape)
        for i in range(total_obs):
            filt_data[i, :] = hpfilter(data[i,:], lamb=1600)[0]

        std = filt_data.std(ddof=1, axis=1).mean()

        filt_df = pd.DataFrame(filt_data).T
        all_corrs = np.zeros(total_obs)
        for i in range(total_obs):
            all_corrs[i] = filt_df[i].autocorr()

        autos = all_corrs.mean()

    else:
        diff = (data[2:] - data[1:-1]) / data[1:-1]
        growth = np.mean(diff)

        filt = hpfilter(np.log(data), lamb=1600)[0]
        std = np.std(filt, ddof=1)
        autos = np.corrcoef(filt[2:], filt[1:-1])[0, 1]
    return [growth, std, autos]

y2_moms = create_moments(y2_data)
y3_moms = create_moments(y3_data)
data_moms = create_moments(output)
all_moms = np.array([y2_moms, y3_moms, data_moms])
moms_df = pd.DataFrame(all_moms)
new_ind = {0: 'y2', 1: 'y3', 2: 'data'}
moms_df.columns = ['mean growth', 'Std. Dev', 'Auto. Corr.']
moms_df = moms_df.rename(index=new_ind)
moms_df
```

Out[2]:

	mean growth	Std. Dev	Auto. Corr.
y2	0.013097	0.035779	0.776455
y3	0.041858	0.036069	0.772666
data	0.007912	0.016807	0.847685

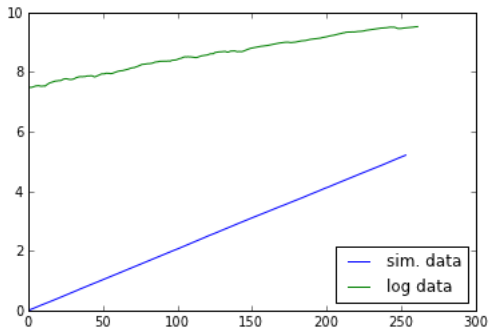
Problem 3

In [3]:
```
def opt_func(x):
    """
    do 50 mc's and return diff between moments and data moments
    """
    p11, p22, g1, g2, sigma = x

    marcov_mat = np.array([[p11, 1 - p11],
                           [1 - p22, p22]], dtype=float).cumsum(1)
    g22 = np.array([g1, g2])
    sims = 10

    y2_sim = np.zeros((sims, T))

    for sim in range(sims):
        s2 = 0
        epsilon = np.random.normal(0, sigma, T)

        for t in range(1, T):
            r_num = np.random.rand()
            y2_sim[sim, t] = g22[s2] + y2_sim[sim, t - 1] + epsilon[t]
            s2 = np.where(marcov_mat[s2, :] > r_num)[0][0]

    sim_moms = np.asarray(create_moments(y2_sim))

    difference = np.linalg.norm(sim_moms - np.asarray(data_moms))

    return difference

x0 = [.6, .3, .02, -.01, .0015]
p11, p22, g1o, g2o, sigmao = fmin(opt_func, x0, xtol=1e-4, maxfun=1e8)
print 'the optimal parameters are p1 = %.3f, p2 = %.3f, g1 = %.3f, g2 = %.3f, sigma = %.3f ' %(p11, p22, g1o, g2o, sigmao)
```

```
Warning: Maximum number of iterations has been exceeded
the optimal parameters are p1 = 0.604, p2 = 0.323, g1 = 0.021, g2 = -0.010, sigma = 0.001
```

Note that because we are trying to optimize over a random process, we will never actually converge. This is pretty close, and probably the best we can hope for without reforming the problem/solution approach a bit. (see plot below for more evidence that we are close)

In [4]:
```
opt_marcov = np.array([[p11, 1 - p11],[1 - p22, p22]]).cumsum(1)
opt_g = np.array([g1o, g2o])
y_opt = np.zeros(254)
s2_opt = 0
opt_eps = np.random.normal(0, sigmao, 254)
for t in range(1, 254):
    y_opt[t] = opt_g[s2_opt] + y_opt[t - 1] + opt_eps[t]
    rand_num = np.random.rand()
    s2 = np.where(opt_marcov[s2_opt, :] > rand_num)[0][0]
pl.plot(range(254), y_opt, label='sim. data')
pl.plot(range(output.size), np.log(output), label='log data')
pl.legend(loc=0)
pl.show()
```



Note that this plot shows that the general shape of the two plots is very similar. The main difference is a scale factor. This is encouraging.