In [30]:
```
from __future__ import division
import numpy as np
import scipy.optimize as opt
import math
import pandas as pd
from numpy.matlib import repmat
import matplotlib.pyplot as plt
```

In [6]:
```
# Here we define the functions necessary to solve the problem.  See Documen
def ksssolve(kvec, params):
    '''
    Used in solving for steady state.  Uses model equations to find the
    Euler Errors and then returns the difference in Euler Errors.

    Parameters
    ----------
    kvec : np.ndarray(1dim)
        1 x 2 vector of capital distribution for current period

    params : np.ndarray(1dim)
        Vector that contains the parameters necessary for calculation

    Returns
    -------
    dif : np.ndarray(1dim)
        Vector that contains the two Euler Errors produced by kvec
    '''

    k2 = kvec[0]
    k3 = kvec[1]

    beta = params[0]
    delta = params[1]
    gamma = params[2]
    A = params[3]
    alpha = params[4]

    L = 2
    K = k2 + k3
    w = (1-alpha) * A * (K/L)**alpha
    r = alpha * A * (L/K)**(1-alpha)

    c1 = w - k2
    c2 = w + (1 + r - delta) * k2 - k3
    c3 = (1 + r - delta) * k3

    MU1 = c1**(-gamma)
    MU2 = c2**(-gamma)
    MU3 = c3**(-gamma)

    Eul1 = MU1 - beta * (1 + r - delta) * MU2
    Eul2 = MU2 - beta * (1 + r - delta) * MU3
    diff = np.array([Eul1, Eul2])
    return diff


def k32solve(k32, params, pathvals):
    '''
    Used in solving for K32.  Uses model equations to find the
    Euler Errors and then returns the difference in Euler Errors.

    Parameters
    ----------
    k32 : float
        intial guess for k32

    params : np.ndarray(1dim)
        Vector that contains the parameters necessary for calculation

    pathvals : np.ndarray(1dim)
        Vector that contains the paths that are established in OLG3per.py

    Returns
    -------
    dif : float
        The Euler Error produced by Euler Equation 2
    '''

    beta = params[0]
    delta = params[1]
    gamma = params[2]
    A = params[3]
    alpha = params[4]

    w1 = pathvals[0]
    r1 = pathvals[1]
    r2 = pathvals[2]
    k21 = pathvals[3]

    c2 = w1 + (1 + r1 - delta) * k21 - k32
    c3 = (1 + r2 - delta) * k32

    MU2 = c2**(-gamma)
    MU3 = c3**(-gamma)

    Eul2 = MU2 - beta * (1 + r2 - delta) * MU3

    diff = Eul2

    return diff


def ktsolve(kvec, params, pathvals):
    '''
    Used in solving for kt.  Uses model equations to find the
    Euler Errors and then returns the difference in Euler Errors.
    Parameters
    ----------
    kvec : np.ndarray(1dim)
        intial guess for k32

    params : np.ndarray(1dim)
        Vector that contains the parameters necessary for calculation

    pathvals : np.ndarray(1dim)
        Vector that contains the paths that are established in OLG3per.py

    Returns
    -------
    dif : np.ndarray(1dim)
        Vector that contains the two Euler Errors produced by kvec
    '''

    k2tp1 = kvec[0]
    k3tp2 = kvec[1]

    beta = params[0]
    delta = params[1]
    gamma = params[2]
    A = params[3]
    alpha = params[4]

    wt = pathvals[0]
    wtp1 = pathvals[1]
    rtp1 = pathvals[2]
    rtp2 = pathvals[3]

    c1 = wt - k2tp1
    c2 = wtp1 + (1 + rtp1 - delta) * k2tp1 - k3tp2
    c3 = (1 + rtp2 - delta) * k3tp2

    MU1 = c1**(-gamma)
    MU2 = c2**(-gamma)
    MU3 = c3**(-gamma)

    Eul1 = MU1 - beta * (1 + rtp1 - delta) * MU2
    Eul2 = MU2 - beta * (1 + rtp2 - delta) * MU3
    diff = np.array([Eul1, Eul2])

    return diff
```

Here we solve for the steady state with the intial β, where $\beta = .96^{20}$

In [9]:
```
#-----------------------------------------------------------------------------#
# Set parameters before we begin
#-----------------------------------------------------------------------------#
# beta  = 20-year discount factor from lifetime utility function
# delta = 20-year depreciation rate of capital investment
# gamma = coefficient of relative risk aversion from CRRA utility function
# A     = productivity parameter from production function
# alpha = capital share of income from production function
# xi    = parameter for convex combination updating of Kpaths in TPI
#-----------------------------------------------------------------------------#
beta = 0.96**20
delta = 1 - (1 - 0.05)**20
gamma = 3
A = 1
alpha = 0.35
xi = 0.25
```

In [11]:
```
#-----------------------------------------------------------------------------#
# Solve for steady-state
#-----------------------------------------------------------------------------#
# kinit   = 1 x 2 vector, initial guess for steady state values of steady-
#           state distribution of capital savings (k_2,k_3)
# params  = 1 x 5 vector of parameters to be passed into the fsolve file
# options = options for fsolve file
# kssvec  = 1 x 2 vector of steady-state equilibrium distribution of
#           capital
# k2ss    = scalar, steady-state savings of young for middle-age
# k3ss    = scalar, steady-state savings of middle-age for old
# Kss     = scalar, steady-state aggregate capital stock
# Lss     = scalar, steady-state aggregate labor demand
# Yss     = scalar, steady-state aggregate output
# wss     = scalar, steady-state real wage
# rss     = scalar, steady-state net interest rate
# c1ss    = scalar, steady-state consumption when young (i=1)
# c2ss    = scalar, steady-state consumption when middle age (i=2)
# c3ss    = scalar, steady-state consumption when old (i=3)
# Css     = scalar, steady-state aggregate consumption
#-----------------------------------------------------------------------------#

kinit = [0.1, 0.1]
params = np.array([beta, delta, gamma, A, alpha])

# options   = optimset('Display','off','MaxFunEvals',100000,..
#                         'MaxIter',1000,'TolFun',1e-15)

kssvec = opt.fsolve(ksssolve, kinit, args=(params), xtol=1e-10)

k2ss = kssvec[0]
k3ss = kssvec[1]
```

```
Kss = k2ss + k3ss
Lss = 2
Yss = A*(Kss**alpha) * (Lss**(1-alpha))
wss = (1-alpha) * A * (Kss/Lss)**alpha
rss = alpha * A * (Lss/Kss)**(1-alpha)
c1ss = wss - k2ss
c2ss = wss + (1 + rss - delta) * k2ss - k3ss
c3ss = (1 + rss - delta) * k3ss
Css = c1ss + c2ss + c3ss
```

Here we solve for steady state when $\beta = .55$

In [12]:
```
beta2 = .55
#---------------------------------------------------------------------------#
# Solve for steady-state
#---------------------------------------------------------------------------#
# kinit   = 1 x 2 vector, initial guess for steady state values of steady-
#           state distribution of capital savings (k_2,k_3)
# params  = 1 x 5 vector of parameters to be passed into the fsolve file
# options = options for fsolve file
# kssvec  = 1 x 2 vector of steady-state equilibrium distribution of
#           capital
# k2ss    = scalar, steady-state savings of young for middle-age
# k3ss    = scalar, steady-state savings of middle-age for old
# Kss     = scalar, steady-state aggregate capital stock
# Lss     = scalar, steady-state aggregate labor demand
# Yss     = scalar, steady-state aggregate output
# wss     = scalar, steady-state real wage
# rss     = scalar, steady-state net interest rate
# c1ss    = scalar, steady-state consumption when young (i=1)
# c2ss    = scalar, steady-state consumption when middle age (i=2)
# c3ss    = scalar, steady-state consumption when old (i=3)
# Css     = scalar, steady-state aggregate consumption
#---------------------------------------------------------------------------#

kinit2 = [0.1, 0.1]
params2 = np.array([beta2, delta, gamma, A, alpha])

# options   = optimset('Display','off','MaxFunEvals',100000,..
#                       'MaxIter',1000,'TolFun',1e-15)

kssvec2 = opt.fsolve(ksssolve, kinit2, args=(params2), xtol=1e-10)

k2ss2 = kssvec2[0]
k3ss2 = kssvec2[1]
Kss2 = k2ss2 + k3ss2
Lss2 = 2
Yss2 = A*(Kss2**alpha) * (Lss2**(1-alpha))
wss2 = (1-alpha) * A * (Kss2/Lss2)**alpha
rss2 = alpha * A * (Lss2/Kss2)**(1-alpha)
c1ss2 = wss2 - k2ss
```

```
c2ss2 = wss2 + (1 + rss2 - delta) * k2ss2 - k3ss2
c3ss2 = (1 + rss2 - delta) * k3ss2
Css2 = c1ss2 + c2ss2 + c3ss2
```

In [35]:

```
# This will solve for problem 3
k21 = .8 * k2ss
k31 = 1.1 * k3ss
K1 = k21 + k31
T = 30

Kpath = np.hstack([np.linspace(K1, Kss, T), np.ones(T)*Kss])
wpath = ((1-alpha) * A) * (Kpath/Lss)**alpha
rpath = (alpha*A) * (Lss/Kpath)**(1-alpha)

i = 0
maxit = 50
dist = 10
toler = 10**(-9)

while dist > toler and i <= maxit:
    i = i + 1

    Knewmat = np.zeros((2*T, 3))
    Knewmat[0, :] = np.array([k21, k31, K1])

    # solve for initial middle age savings decision k_{3,2}
    k32init = 0.01
    pathvals = np.array([wpath[0], rpath[0], rpath[1], k21])
    k32 = opt.fsolve(k32solve, k32init, args=(params, pathvals), xtol=1e-10
    Knewmat[1, 1] = k32

    # Solve 2-period problems for rest of households
    for time in xrange(2*T-2):
        kinits = [.01, .01]
        pathvals = np.array([wpath[time], wpath[time+1],
                             rpath[time+1], rpath[time+2]])
        kvec = opt.fsolve(ktsolve, kinits, args=(params, pathvals), xtol=1e
        Knewmat[time+1, 0] = kvec[0]
        Knewmat[time+2, 1] = kvec[1]
        Knewmat[time+1, 2] = Knewmat[time+1, 0] + Knewmat[time+1, 1]

    dist = sum((Knewmat[0: T+5, 2] - Kpath[0: T+5])**2)
    Kpath = xi * Knewmat[0: T+5, 2] + (1-xi) * Kpath[0: T+5]
    Kpath = np.hstack([Kpath, Kss * np.ones(T-5)])
    wpath = ((1-alpha) * A) * (Kpath/Lss)**alpha
    rpath = (alpha * A) * (Lss/Kpath)**(1-alpha)
```

```
In [39]: (Kpath).size
```

```
Out[39]: 60
```

Problem 1: The steady-state equilibrium with the initial parameter values are listed here.

Steady State Consumption levels

$\bar{C_1} = .2140$

$\bar{C_2} = .2227$

$\bar{C_3} = .2318$

Steady State Capital levels

$\bar{K_2} = .0286$

$\bar{K_3} = .0909$

Steady State Wage and Interest Rate

$\bar{W} = .2421$

$\bar{r} = 2.1916$

Problem 2: The steady-state equilibrium with the initial parameter values are listed here.

Steady State Consumption levels

$\bar{C_1} = .2396$

$\bar{C_2} = .2404$

$\bar{C_3} = .2552$

Steady State Capital levels

$\bar{K_2} = .0413$

$\bar{K_3} = .1173$

Steady State Wage and Interest Rate

$\bar{W} = .2677$

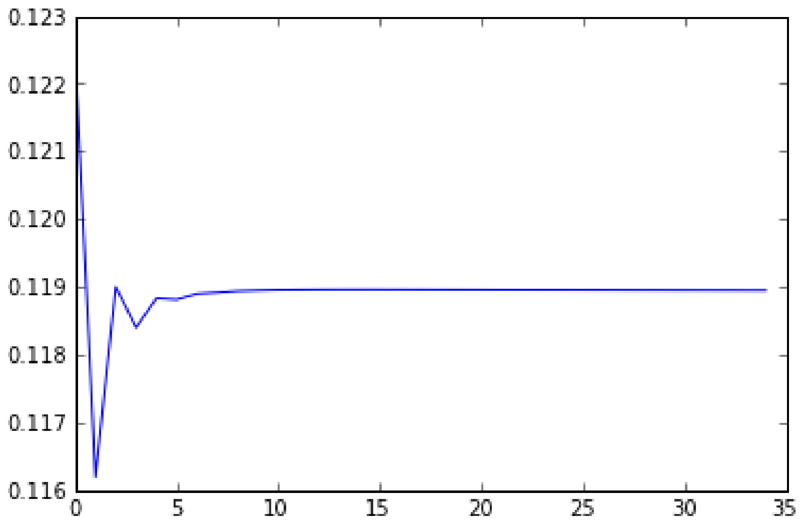
$\bar{r} = 1.8179$

We see that the consumption goes down in early periods of life and that investment in capital goes up. This makes sense because as people become more patient then they are more willing to wait to consume their goods.

Problem 3: The TPI was done using the code above.

Problem 4: The equilibrium time path is plotted below coming from $.8k_2$ and $1.1k_3$ and moves to the steady state.

In [22]:
```
plt.plot(Kpath[0:T+5])
plt.show()
```



It takes the capital path 6 periods to be within .0001 of the steady state.

In [47]:
```
df = pd.DataFrame({'Kpath':Kpath, 'Kpath- ss':Kpath- Kss}, index=range(1, K
df.index.name = 'time'
df
```

Out[47]:

	Kpath	Kpath-ss
time		
1	0.122427	0.003478
2	0.116197	-0.002752
3	0.119004	0.000055
4	0.118401	-0.000548
5	0.118834	-0.000115
6	0.118815	-0.000134
7	0.118901	-0.000048
8	0.118918	-0.000031
9	0.118940	-0.000009
10	0.118949	0.000000
11	0.118956	0.000007
12	0.118959	0.000010
13	0.118961	0.000012
14	0.118962	0.000013

15	0.118962	0.000013
16	0.118962	0.000012
17	0.118961	0.000012
18	0.118961	0.000012
19	0.118960	0.000011
20	0.118959	0.000010
21	0.118959	0.000010
22	0.118958	0.000009
23	0.118958	0.000008
24	0.118957	0.000008
25	0.118956	0.000007
26	0.118956	0.000006
27	0.118955	0.000006
28	0.118954	0.000005
29	0.118954	0.000005
30	0.118953	0.000004
31	0.118952	0.000003
32	0.118952	0.000003
33	0.118951	0.000002
34	0.118951	0.000002
35	0.118950	0.000001
36	0.118949	0.000000
37	0.118949	0.000000

```
In [ ]:
```