In [137]:
```
import numpy as np
import numpy.random as npr
import scipy.stats as scis
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [379]:
```
# Initialize parameters
R = 1.05
beta = 0.93
abar = -1
ymean = 10
```

In [380]:
```
# create a sequence of y, a, and V
n = 100
y_min = 0.01
y_max = 20.01
ylist = np.linspace(y_min, y_max, n)
m = 100
a_min = abar
a_max = 5
alist = np.linspace(a_min,a_max,m)
```

In [140]:
```
# Define a guessed form of value function
def v(a,y):
    return np.log(a + y - abar)
# Define utility function
def u(c):
    return np.log(c)
```

In [340]:
```
# set seed
npr.seed(233)

# Define the probability transition matrix
def transmatrix(n, m, var):
    mat = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            mat[i][j] = scis.norm.cdf(ylist[j] - 0.5 * ylist[i] - ymean/
2 + y_max/(2 * n), loc = 0, scale = var) - scis.norm.cdf(ylist[j] - 0.5
* ylist[i] - ymean/2 - y_max/(2 * n),loc = 0, scale = var)

    return mat
```

In [343]:
```
mat = transmatrix(n, m, 1)
print(sum(mat[1]))
```

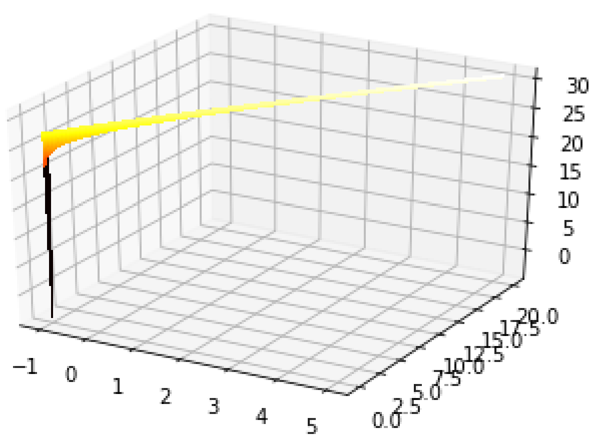
```
0.9904948997969462
```

In [335]:
```
# initialize guesses
tol = 0.01
Tv = np.zeros(len(ylist)) + 20
a = 2
ap = 1
y = 0.01


# initial guess for value assuming the functional form of log
v_0_guess = np.zeros((m,n))
for i in range(m):
    for j in range(n):
        v_0_guess[i][j] = np.log(alist[i] + ylist[j] - abar)
# initial guess
```

In [296]:
```
# loop ver it

# store the a' that maximizes value function given a and y
ap_matrix = np.zeros((m, n))

# store the current maximum value function corresponding to the argmax
 a' given a and y


def bellman(V0, matrix, R):
    Tv = V0
    ap = 0
    for i in range(m):
        a = alist[i]

    # temporary list that stores the v(a') given a and y for each a'. we
use this to find the max v and argmax a'
        vals = np.zeros(m)
        for j in range(n):
            y = ylist[j]
            prob = matrix[j,:]

        # use this as temporary value variable to find the maximum value
            vmax = 0
            for k in range(m):
                if R * alist[i] - alist[k] + ylist[j] <= 0:
                    continue
                else:
                # the guessed maximum value from last iteration
                    v_guess_list = Tv[k,:]
                    val = u(R * alist[i] - alist[k] + ylist[j]) + beta *
np.dot(v_guess_list, matrix[j,:])
                    if val > vmax:
                        vmax = val
                        ap = alist[k]
                Tv[i][j] = vmax
                ap_matrix[i][j] = ap
    return (Tv, ap_matrix)

        #Tv[i][j] = (float(max(vals)[0]))
        #ap_matrix[i][j] = ap
        #v_new = u(R * alist[i] - ap_matrix[i][j] + ylist[j]) + beta * n
p.dot(Tv, matrix[j,:])

        #v_new_list.append((v_new, np.dot(Tv, matrix[j,:]), alist[i], a
p, y))
        #if abs(v_new - np.dot(Tv, prob)) < tol:
        #    break
```

In [348]:
```
def iteration(V0, periods, tol, matrix, R):
    for i in range(periods):
        temp = V0
        V_next = bellman(V0, matrix, R)[0]
        dif = np.abs(V_next - temp)
        dif = np.reshape(dif, (m * n))
        dif = max(dif)
        if dif < tol:
            ap = bellman(V0, matrix, R)[1]
            break
        else:
            V0 = V_next
            continue
    return (V_next, ap)
```

In [147]:
```
X = alist
Y = ylist
Z = Tv
# plot the Tv matrix
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot', linewi
dth=0, antialiased=False)
```



In [166]:
```
Tv = iteration(v_0_guess, 100, 0.001)[0]
print(Tv)
```

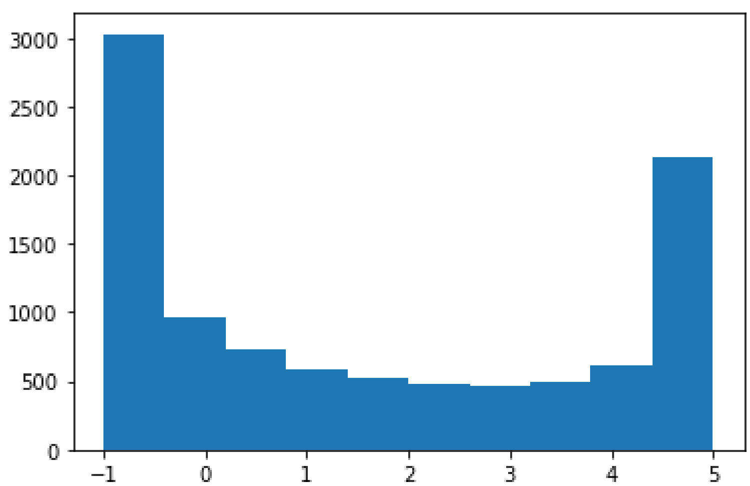
```
0.0
[[-4.60517019 23.84612346 24.68490471 ... 30.5614532  30.58832536
  30.61497626]
 [21.90027985 24.18610903 24.85459016 ... 30.56584108 30.59266567
  30.61926931]
 [23.20653151 24.43453458 24.99330413 ... 30.57021157 30.59699412
  30.62355652]
 ...
 [27.47685927 27.5399637  27.60159875 ... 30.93462909 30.95742777
  30.98006936]
 [27.4883079  27.55112829 27.61246599 ... 30.93802209 30.96075912
  30.98334034]
 [27.49971074 27.5621979  27.62324287 ... 30.94267233 30.96540194
  30.98797586]]
```

In [167]:
```
ap = iteration(v_0_guess, 100, 0.001)[1]
```

```
0.0
```

In [168]:
```
counts, bins = np.histogram(ap)
plt.hist(bins[:-1], bins, weights=counts)
```

Out[168]:
```
(array([3029.,  967.,  724.,  576.,  519.,  482.,  462.,  489.,  614.,
        2138.]),
 array([-1. , -0.4,  0.2,  0.8,  1.4,  2. ,  2.6,  3.2,  3.8,  4.4,  5.
]),
 <a list of 10 Patch objects>)
```

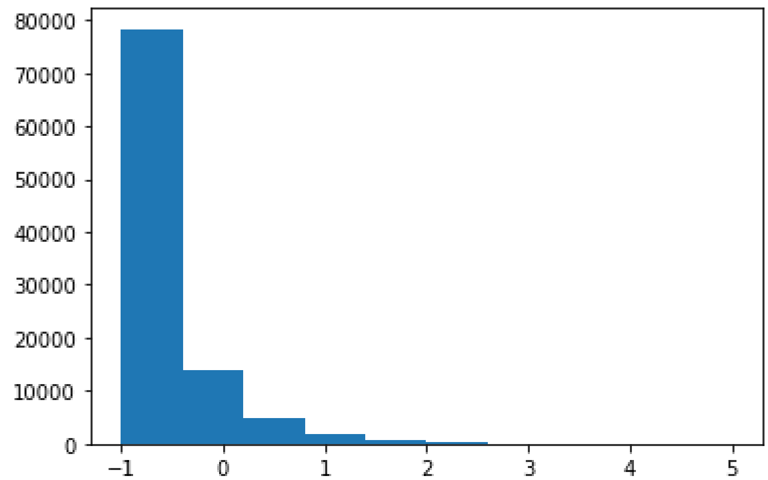


In [313]:
```
# simulate data

def simulate(sim_n, y0, a0, var, ap):
    npr.seed(233)
    sim_matrix = np.zeros((sim_n, 2))

    for i in range(sim_n):
        a_cur = a0
        y_cur = y0

        # find the position of y_cur and a_cur
        dif_a = abs(alist - a0)
        min_dif1 = min(dif_a)
        a_pos = np.where(dif_a==min_dif1)[0][0]
        a_p = alist[a_pos]
        sim_matrix[i][0] = a_p

        dif_y = abs(ylist - y0)
        min_dif2 = min(dif_y)
        y_pos = np.where(dif_y == min_dif2)[0][0]
        y_p = ylist[y_pos]
        sim_matrix[i][1] = y_p

        a_next = ap[a_pos][y_pos]
        y_next = 0.5 * y_p + 5 + npr.normal(0, var)

        a0 = a_next
        y0 = y_next
    return sim_matrix
```

In [270]:
```
sim = simulate(100000, 0.01, 0)
```

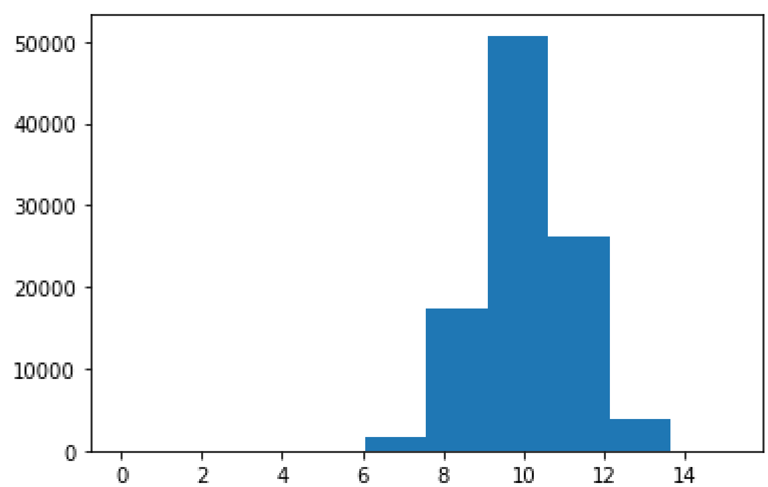
In [271]:
```
counts, bins = np.histogram(sim[:,0])
plt.hist(bins[:-1], bins, weights=counts)
```

Out[271]:
```
(array([7.8300e+04, 1.3777e+04, 4.9740e+03, 1.8580e+03, 7.0700e+02,
        2.4100e+02, 8.5000e+01, 4.4000e+01, 1.0000e+01, 4.0000e+00]),
 array([-1. , -0.4,  0.2,  0.8,  1.4,  2. ,  2.6,  3.2,  3.8,  4.4,  5.
]),
 <a list of 10 Patch objects>)
```



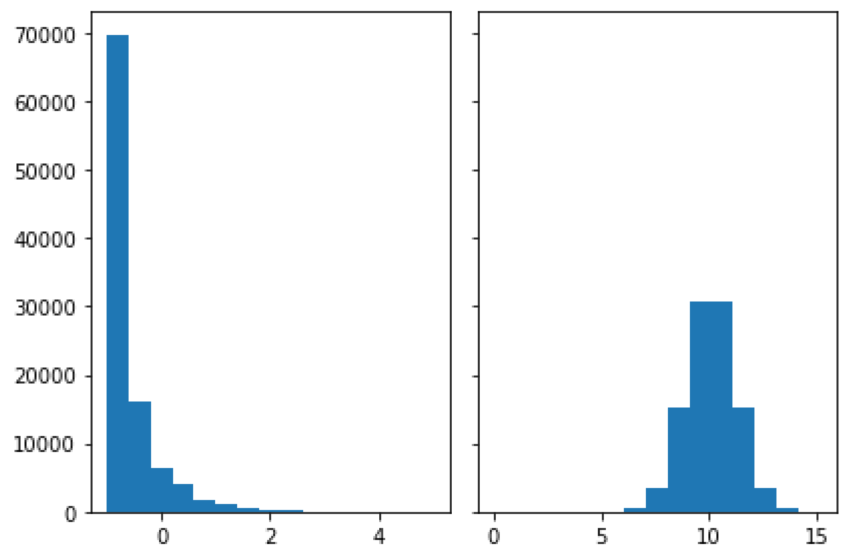
In [272]:
```
counts, bins = np.histogram(sim[:,1])
plt.hist(bins[:-1], bins, weights=counts)
```

Out[272]:
```
(array([1.0000e+00, 0.0000e+00, 1.0000e+00, 2.2000e+01, 1.7630e+03,
        1.7396e+04, 5.0754e+04, 2.6080e+04, 3.9150e+03, 6.8000e+01]),
 array([1.00000000e-02, 1.52515152e+00, 3.04030303e+00, 4.55545455e+00,
        6.07060606e+00, 7.58575758e+00, 9.10090909e+00, 1.06160606e+01,
        1.21312121e+01, 1.36463636e+01, 1.51615152e+01]),
 <a list of 10 Patch objects>)
```



In [273]:
```
n_bins = 15
fig, axs = plt.subplots(1, 2, sharey=True, tight_layout=True)
x = sim[:,0]
y = sim[:,1]
# We can set the number of bins with the `bins` kwarg
axs[0].hist(x, bins=n_bins)
axs[1].hist(y, bins=n_bins)
```

Out[273]:
```
(array([1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
        2.2000e+01, 4.2800e+02, 3.6060e+03, 1.5125e+04, 3.0871e+04,
        3.0837e+04, 1.5126e+04, 3.5460e+03, 4.2200e+02, 1.5000e+01]),
 array([1.00000000e-02, 1.02010101e+00, 2.03020202e+00, 3.04030303e+00,
        4.05040404e+00, 5.06050505e+00, 6.07060606e+00, 7.08070707e+00,
        8.09080808e+00, 9.10090909e+00, 1.01110101e+01, 1.11211111e+01,
        1.21312121e+01, 1.31413131e+01, 1.41514141e+01, 1.51615152e+0
1]),
 <a list of 15 Patch objects>)
```



In [274]:
```
# the percentage that agents reach the lower bound
sum(sim[:,0] == -1)/100000
```

Out[274]: 0.15386

In [275]:
```
# average saving
sum(sim[:,0])/len(sim[:,0])
```

Out[275]: -0.6261084848479859

In [325]:
```
start = timeit.default_timer()
# change transition probability
matrix2 = matrix(n, m, 1.5)
stop = timeit.default_timer()

print('Time: ', stop - start)
```

Time: 1.8006787569975131

In [295]:
```
print(matrix2)
```

```
[[2.09595923e-04 3.25039845e-04 4.95021874e-04 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [1.67169175e-04 2.61603148e-04 4.02034889e-04 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [1.32728158e-04 2.09595923e-04 3.25039845e-04 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 ...
 [4.33641118e-23 1.61883155e-22 5.93479970e-22 ... 3.18191986e-04
  2.04996336e-04 1.29699101e-04]
 [2.22917824e-23 8.39748833e-23 3.10661340e-22 ... 3.93741382e-04
  2.55976983e-04 1.63427367e-04]
 [1.14075512e-23 4.33641118e-23 1.61883155e-22 ... 4.85027531e-04
  3.18191986e-04 2.04996336e-04]]
```

In [360]:
```
import timeit

start = timeit.default_timer()

#Your statements here



V2 = bellman(v_0_guess, matrix2, 1.05)[0]
V2 = iteration(V2, 1000, 0.001, matrix2, R)[0]
a2 = iteration(V2, 1000, 0.001, matrix2, R)[1]

stop = timeit.default_timer()

print('Time: ', stop - start)
```

```
Time:  24.797642481993535
```

In [350]:
```
print(V2)
```

```
[[ 0.69813472 23.70683873 24.549535   ... 30.48471668 30.50999396
  30.53461928]
 [21.75754823 24.04793819 24.72033372 ... 30.48910635 30.51434122
  30.53892395]
 [23.06379989 24.29636374 24.85904769 ... 30.49348585 30.51867503
  30.54341272]
 ...
 [27.34871231 27.41523634 27.47977188 ... 30.85862813 30.8798541
  30.90048768]
 [27.36045933 27.42664352 27.49089704 ... 30.86263023 30.88379419
  30.90436697]
 [27.37215667 27.4380052  27.50190816 ... 30.86738861 30.88854473
  30.90910972]]
```

In [361]:
```
# simulate the data when var = 1.5
import timeit

start = timeit.default_timer()

#Your statements here


sim2 = simulate(10000, 0.01, 0, 1.5, a2)

stop = timeit.default_timer()

print('Time: ', stop - start)
```

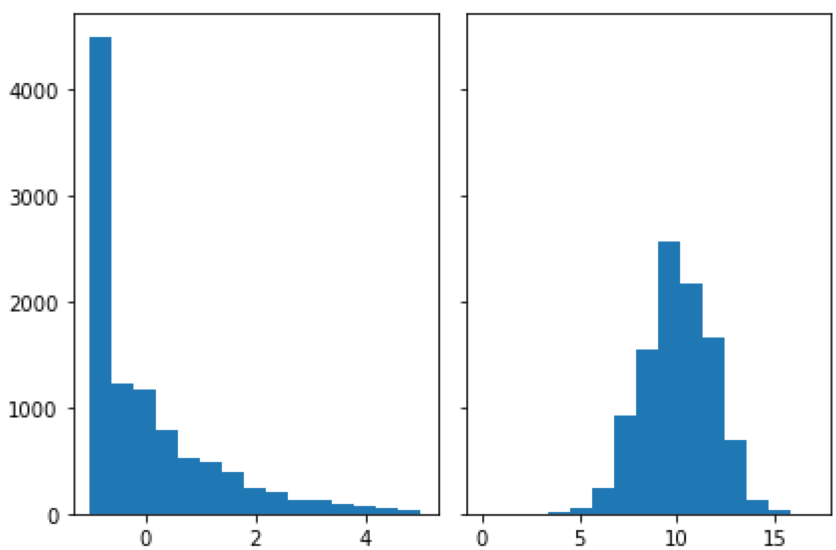
```
Time:  0.36281976000464056
```

In [362]:
```
sim2
```

Out[362]:
```
array([[-0.03030303,  0.01      ],
       [-1.        ,  6.27262626],
       [-1.        ,  8.49484848],
       ...,
       [-0.75757576,  8.69686869],
       [-0.6969697 ,  8.69686869],
       [-0.63636364,  9.7069697 ]])
```

In [363]:
```
n_bins = 15
fig, axs = plt.subplots(1, 2, sharey=True, tight_layout=True)
x = sim2[:,0]
y = sim2[:,1]
# We can set the number of bins with the `bins` kwarg
axs[0].hist(x, bins=n_bins)
axs[1].hist(y, bins=n_bins)
```

Out[363]:
```
(array([1.000e+00, 1.000e+00, 0.000e+00, 6.000e+00, 4.500e+01, 2.330e+0
2,
        9.180e+02, 1.535e+03, 2.570e+03, 2.161e+03, 1.665e+03, 6.960e+0
2,
        1.350e+02, 3.000e+01, 4.000e+00]),
 array([1.00000000e-02, 1.14131313e+00, 2.27262626e+00, 3.40393939e+00,
        4.53525253e+00, 5.66656566e+00, 6.79787879e+00, 7.92919192e+00,
        9.06050505e+00, 1.01918182e+01, 1.13231313e+01, 1.24544444e+01,
        1.35857576e+01, 1.47170707e+01, 1.58483838e+01, 1.69796970e+0
1]),
 <a list of 15 Patch objects>)
```



In [364]:
```
# average saving under a high varaince of y
sum(sim2[:,0])/len(sim2[:,0])
```

Out[364]: 0.006466666666667196

In [365]:
```
# the percentage that agents reach the lower bound under variance of 1.5
sum(sim2[:,0] == -1)/100000
```

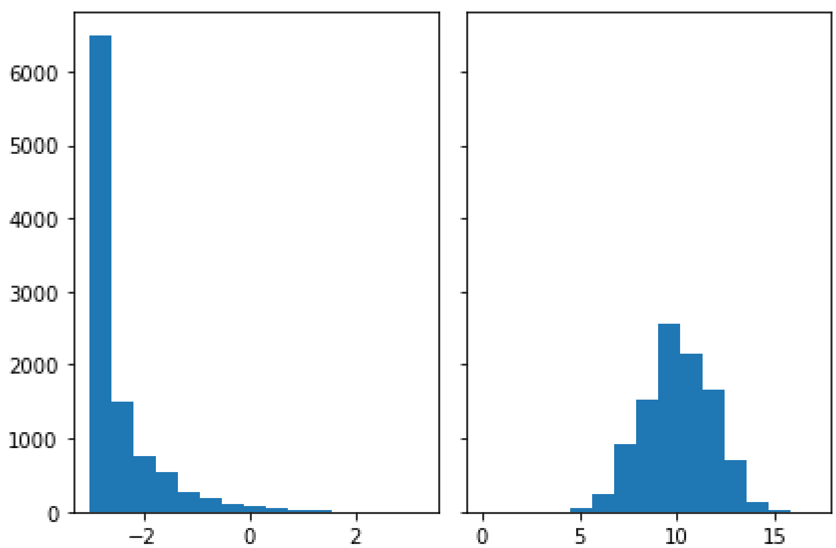
Out[365]: 0.01763

In [366]:
```
# change abar to -3

alist = np.linspace(-3,5,100)

V3 = bellman(v_0_guess, mat, 1)[0]
V3 = iteration(V3, 1000, 0.001, mat, R)[0]
a3 = iteration(V3, 1000, 0.001, mat, R)[1]
```

In [367]:
```
sim3 = simulate(10000, 0.01, 0, 1.5, a3)
```

In [368]:
```
n_bins = 15
fig, axs = plt.subplots(1, 2, sharey=True, tight_layout=True)
x = sim3[:,0]
y = sim3[:,1]
# We can set the number of bins with the `bins` kwarg
axs[0].hist(x, bins=n_bins)
axs[1].hist(y, bins=n_bins)
```

Out[368]:
```
(array([1.000e+00, 1.000e+00, 0.000e+00, 6.000e+00, 4.500e+01, 2.330e+0
2,
        9.180e+02, 1.535e+03, 2.570e+03, 2.161e+03, 1.665e+03, 6.960e+0
2,
        1.350e+02, 3.000e+01, 4.000e+00]),
 array([1.00000000e-02, 1.14131313e+00, 2.27262626e+00, 3.40393939e+00,
        4.53525253e+00, 5.66656566e+00, 6.79787879e+00, 7.92919192e+00,
        9.06050505e+00, 1.01918182e+01, 1.13231313e+01, 1.24544444e+01,
        1.35857576e+01, 1.47170707e+01, 1.58483838e+01, 1.69796970e+0
1]),
 <a list of 15 Patch objects>)
```



In [374]:
```
# the percentage that people reach the lower bound to -3
x = sim3[:,0]
sum(x==-3)/len(x)
```

Out[374]: 0.2077

In [377]:
```
# I use time it in python and
time = 1.8006787569975131 + 24.797642481993535 + 0.36281976000464056
print((time ,"seconds"))
```

```
(26.96114099899569, 'seconds')
```

In [409]:
```
# now let's find equilibrium R
Rn = 20
Rlist = np.linspace(1.06734578, 1/beta, Rn)
def findR(Rn, tol):
    for i in range(Rn):
        V = bellman(v_0_guess, mat, 1)[0]
        V = iteration(V, 100, 0.001, mat, Rlist[i])[0]
        a = iteration(V, 100, 0.001, mat, Rlist[i])[1]
        sim = simulate(1000, 0.01, 0, 1, a)
        if abs(sum(sim[:,0])/len(sim[:,0])) > tol:
            continue
        else:
            break
    return Rlist[i]
```

In [410]:
```
print(Rlist)
```

```
[1.06734578 1.06776278 1.06817978 1.06859679 1.06901379 1.06943079
 1.06984779 1.07026479 1.0706818  1.0710988  1.0715158  1.0719328
 1.0723498  1.07276681 1.07318381 1.07360081 1.07401781 1.07443481
 1.07485182 1.07526882]
```

In [412]:
```
# This should take 20 * 26.9 =
start = timeit.default_timer()

R_equi = findR(Rn, 0.01)

stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Time:  429.45505577600125
```

In [413]:
```
R_equi
```

Out[413]:
```
1.0736008093718166
```