(a) Equal lamp powers.

In [11]:
```
from math import log
```

In [29]:
```
res = -1
f = []
for y in np.arange(0.0001, 1, 0.0001):
    for row in A:
        res = max(abs(log(np.dot(row,np.ones(10,)*y))), res)
    f.append(res)
    res = -1
%time
```

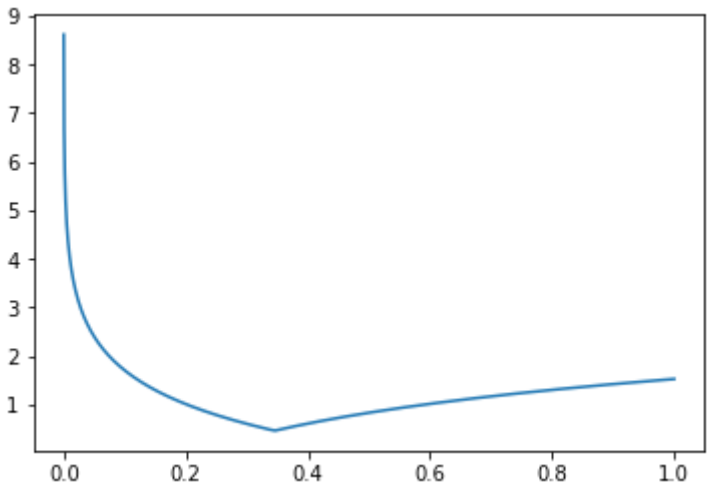
```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.87 µs
```

In [30]:
```
len(f)
```

Out[30]:
```
9999
```

In [31]:
```
plt.plot(np.arange(0.0001, 1, 0.0001), f)
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x116250ef0>]
```



In [32]:
```
print(np.argmin(f)/10000)
np.min(f)
```

```
0.3453
```

Out[32]:
```
0.46767995266043966
```

(b) Least-squares with saturation.

In [33]:
```
A_ = pinv(A)
p2 = np.dot(A_,np.ones((20,)))
```

In [34]:
```
p2[p2 > 1] = 1
p2[p2 < 0] = 0
```

In [35]:
```
p2
```

Out[35]:
```
array([ 1.,  0.,  1.,  0.,  0.,  1.,  0.,  1.,  0.,  1.])
```

In [39]:
```
res = -1
for row in A:
    res = max(abs(log(np.dot(row,p2))), res)
print(res)
```

```
0.8627835570819031
```

(c) Regularized least squares.

In [34]:
```
rho = 0.219
```

In [35]:
```
# rho = 0.1
# while(True):
#     temp = np.dot(pinv(A3), np.vstack((np.ones((20,1)), (rho**0.5)*0.5*np.
#     if temp.min() >= 0 and temp.max() <= 1 :
#         print(rho);
#         break
#     else:
#         rho += 0.001
# print(rho)
# %time
```

In [41]:
```
rho = 0.2190
A3 = np.vstack((A, (rho**0.5)*np.eye(10)))
p3 = (np.dot(pinv(A3), np.vstack((np.ones((20,1)), (rho**0.5)*0.5*np.ones((1
p3
```

Out[41]:
```
array([[  5.00419805e-01],
       [  4.77688566e-01],
       [  8.33041787e-02],
       [  2.25265686e-04],
       [  4.56080593e-01],
       [  4.35428383e-01],
       [  4.59707916e-01],
       [  4.30715783e-01],
       [  4.03431101e-01],
       [  4.52643679e-01]])
```

In [42]:
```
res = -1
for row in A:
    res = max(abs(log(np.dot(row,p3))), res)
print(res)
```

```
0.4438989791535322
```

(d) Chebyshev approximation.

In [44]:
```
from cvxpy import *
```

In [45]:
```
b = np.ones((20,1))
p4 = Variable(10)
objective = Minimize(norm(A*p4 - b, "inf"))
constraints = [0 <= p4, p4 <= 1]
prob = Problem(objective, constraints)
# The optimal objective is returned by prob.solve().
result = prob.solve()
# The optimal value for x is stored in x.value.
print(p4.value)
p4 = p4.value
```

```
[[  9.99999999e-01]
 [  1.16498941e-01]
 [  1.80418261e-11]
 [  1.66764395e-11]
 [  9.99999997e-01]
 [  2.01763782e-09]
 [  1.00000000e+00]
 [  2.49006783e-02]
 [  2.42985357e-10]
 [  9.99999999e-01]]
```

In [12]:
```
p4 = np.array( [[  9.99999999e-01],
 [  1.16498941e-01],
 [  1.80418261e-11],
 [  1.66764395e-11],
 [  9.99999997e-01],
 [  2.01763782e-09],
 [  1.00000000e+00],
 [  2.49006783e-02],
 [  2.42985357e-10],
 [  9.99999999e-01]])
res = -1
for row in A:
    res = max(abs(log(np.dot(row,p4))), res)
print(res)
```

```
0.419824197340787
```

(e) Exact solution.

In [39]:
```
b = np.ones((20,1))
p5 = Variable(10)
#f0 = np.vstack((A*p5, inv_pos(A*p5)))
```

In [40]:
```
objective = Minimize(max_elemwise(max_entries(A*p5),max_entries(inv_pos(A*p5
constraints = [0 <= p5, p5 <= 1]
prob = Problem(objective, constraints)
# The optimal objective is returned by prob.solve().
result = prob.solve()
# The optimal value for x is stored in x.value.
print(p5.value)
```

```
[[  1.00000000e+00]
 [  2.02299180e-01]
 [  3.56868941e-11]
 [  1.40294607e-11]
 [  9.99999997e-01]
 [  2.32004012e-09]
 [  1.00000000e+00]
 [  1.88156919e-01]
 [  1.48570996e-10]
 [  1.00000000e+00]]
```

In [13]:
```
p5 = np.array( [[  1.00000000e+00],
 [  2.02299180e-01],
 [  3.56868941e-11],
 [  1.40294607e-11],
 [  9.99999997e-01],
 [  2.32004012e-09],
 [  1.00000000e+00],
 [  1.88156919e-01],
 [  1.48570996e-10],
 [  1.00000000e+00]])
res = -1
for row in A:
    res = max(abs(log(np.dot(row,p5))), res)
print(res)
```

```
0.35747432077682445
```

In [ ]: