Homework 3: Question 2

In [2]:
```
from ggplot import *
from collections import Counter
from pandas import *
```

Generate 100000 random numbers drawn from PowerLaw(x_min,alpha), given a random number uniformly drawn from [0,1). In this case, $x_{min} = 1, \alpha = 2$. So the formula derived in class becomes:

$$x = \frac{1}{1-r}$$

In [3]:
```
def gen_plaw_nums(N=100000):
    l = []
    for i in xrange(N):
        r = np.random.rand()
        x = round(1/(1-r))
        l.append(x)
    return l
```

Generate empirical histogram from the data.

In [4]:
```
def gen_hist(data,N=1.0e5,filt=False):
    counter = Counter(data)
    if filt: del counter[1]
    x,y = zip(*(counter.items()))
    y = [i/N for i in y]
    return np.array(x),np.array(y)
```

Calculate the empirical CCDF from the data.

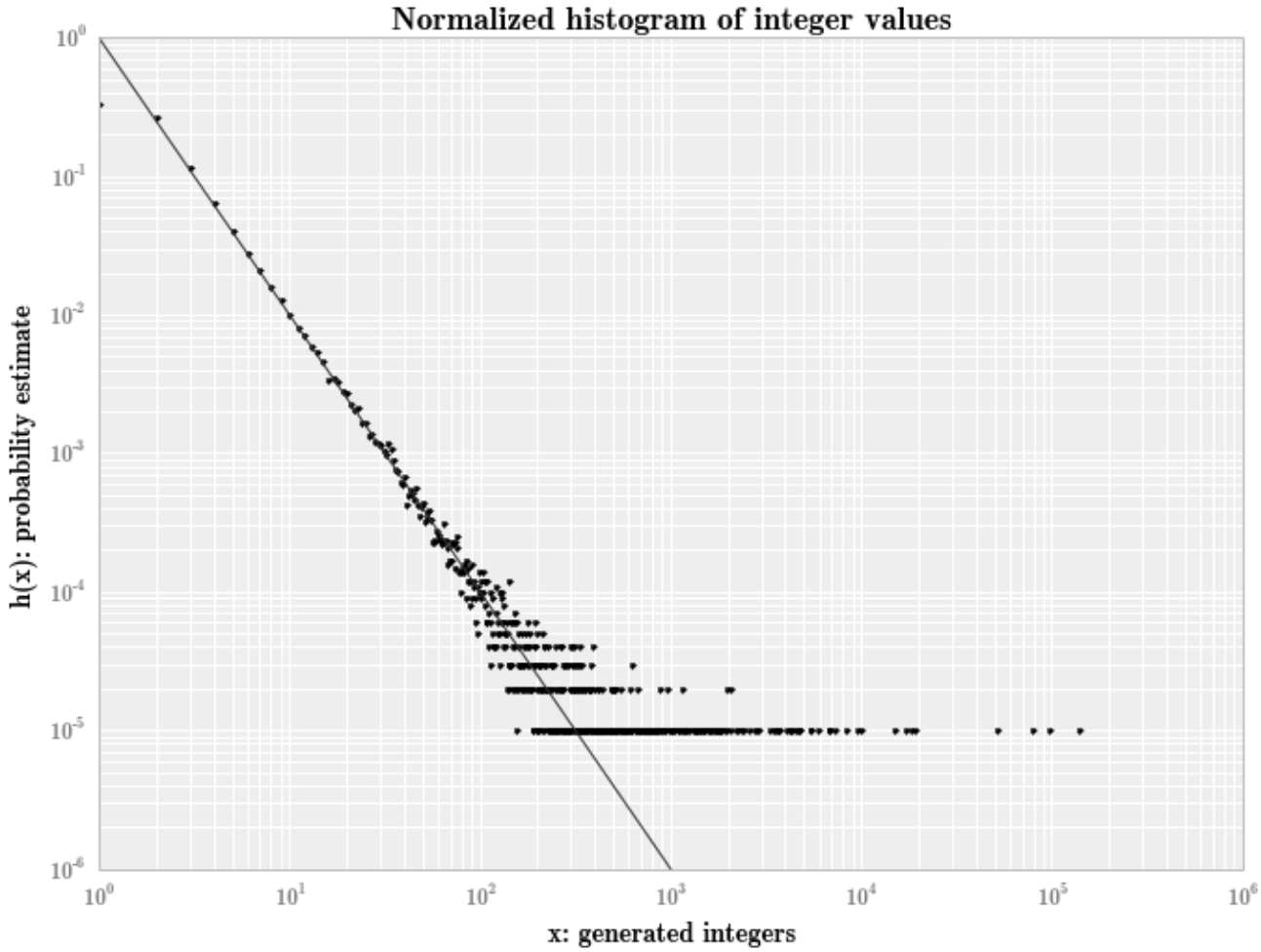
In [5]:
```
def gen_ccdf(data,N=1.0e5):
    counter = Counter(data)
    cum = 0
    ccdf = {}
    for x in sorted(counter.keys(),reverse=True):
        cum += counter[x]
        ccdf[x] = cum/N
    x,y = zip(*(ccdf.items()))
    return np.array(x),np.array(y)
```

In [6]:
```
raw_data = gen_plaw_nums()
```

Q2b) Plot histogram

In [7]:
```
def plot_plaw_hist(data):
    x,y = gen_hist(data)
    plt.loglog(x,y,'k.')
    lx = sorted(x)
    ly = [1/(x**2) for x in lx]
    plt.loglog(lx,ly)
    plt.ylim(ymin=1e-6)
    plt.xlabel("x: generated integers")
    plt.ylabel("h(x): probability estimate")
    plt.title("Normalized histogram of integer values")
```

In [9]:
```
plot_plaw_hist(raw_data)
```



Q2c) Least squares regression on histogram

In [10]:
```
def lst_sq_alpha_hist(data):
    x,y = gen_hist(data)
    X = array([np.log(x), ones(len(x))]).T
    w = np.linalg.lstsq(X,np.log(y))
    return -w[0][0]
```

In [11]:
```
lst_sq_alpha_hist(raw_data)
```

Out[11]:
```
0.95042400890729262
```

Improve by not using all of the data. In particular, skip high values of x. Let's try excluding all $x > 100$.

In [12]:
```
def lst_sq_alpha_hist(data,upper=100):
    data = filter(lambda x: x < upper,data)
    x,y = gen_hist(data,filt=True)
    X = array([np.log(x), ones(len(x))]).T
    w = np.linalg.lstsq(X,np.log(y))
    return -w[0][0]
```

In [13]:
```
lst_sq_alpha_hist(raw_data)
```

Out[13]:
```
2.049405844972978
```

Q2d) Least Squares Regression on CCDF

In [14]:
```
def lst_sq_alpha_ccdf(data):
    x,y = gen_ccdf(data)
    X = array([ np.log(x), ones(len(x))]).T
    w = np.linalg.lstsq(X,np.log(y))
    return (1-w[0][0])
```

In [15]:
```
lst_sq_alpha_ccdf(raw_data)
```

Out[15]:
```
1.9925486191908068
```

Q2e) MLE on data

In [16]:
```
def mle_alpha(data,N=1.0e5):
    logsum = 0.0
    for d in data:
        logsum += log(d)
    return (1+N/logsum)
```

In [17]:
```
mle_alpha(gen_plaw_nums())
```

Out[17]: 2.0405933631653976

Q2f) Compare the three methods

In [18]:
```
alphas_hist = []
alphas_ccdf = []
alphas_mle  = []
for i in xrange(100):
    data = gen_plaw_nums(100000)
    alphas_hist.append(lst_sq_alpha_hist(data))
    alphas_ccdf.append(lst_sq_alpha_ccdf(data))
    alphas_mle.append(mle_alpha(data))
```

In [19]:
```
print np.mean(alphas_hist),np.var(alphas_hist)
```

```
2.02203580048 0.000423750970963
```

In [20]:
```
print np.mean(alphas_ccdf),np.var(alphas_ccdf)
```

```
1.98972120783 0.0012376088257
```

In [21]:
```
print np.mean(alphas_mle),np.var(alphas_mle)
```

```
2.04724378002 1.55176026014e-05
```

In [19]:
```

```