Monte Carlo Estimation

Suppose we have a coin whose probability of "heads" is *p*. *q* is another probability, greater than *p*

Compute the Monte-Carlo estimate of the probability that the number of "heads" in *n* tosses of the coin is at least *n* * *q*

In [4]:
```
from random import random
from pylab import *

n=100   # length of sequence
m=10000 # number of trials
p=0.5   # the true probability
q=0.6   # the hypothesized (or model) probability

count=0
for j in range(m): # Equivalent to java: for(j=0, j<m, j++) {
    # using python list comprehension create a list of length n
    # where each entry is 1 with probability p and 0 with probability 1-p
    outcomes=[(1 if random()<p else 0) for i in range(n)]
    # Check if the number of 1's in outcomes is larger or equal to n*q, if so, add 1 to count
    if sum(outcomes)>=int(n*q): count += 1
# Print a monte-carlo estimate of the probability that the number of 1's in outcomes is at least n*q
print (count+0.0)/m
```

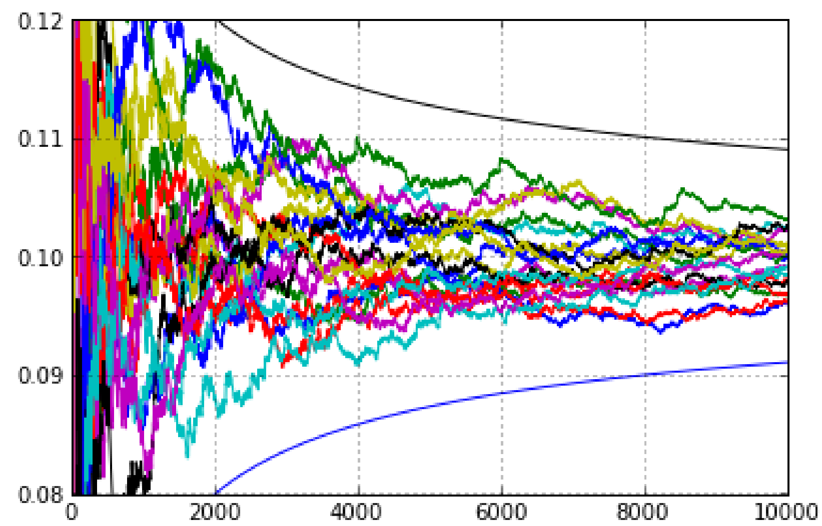
```
0.028
```

The average converges to the mean

Here we generate sequences of length *n* as above and plot the running average of the sequence.
Which means the average of the first *i* elements for each *i*
We then plot an envelope around the mean which contains the sequences with high probability.
The envelope is drawn 3 standard deviations from the mean.

This demonstrates the convergence of the average value to the mean, also known as *The law of large numbers*

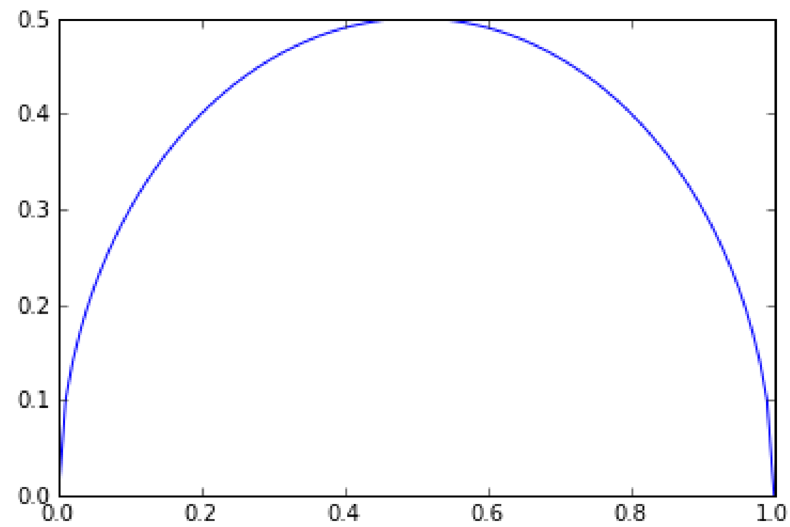
In [5]:
```
n=10000 # length of sequence
m=20    # number of trials
p=0.1   # the true probability

count=0
for j in range(m):
    outcomes=[(1 if random()<p else 0) for i in range(n)]
    cs=cumsum(outcomes)
    run_aver=[cs[i]/(i+1.0) for i in range(len(cs))]
    plot(run_aver)

var=p*(1-p)   # The variance of a binary variable with P(1)=p
upper=[p+3*sqrt(var/(i+1.0)) for i in range(n)]
lower=[p-3*sqrt(var/(i+1.0)) for i in range(n)]
plot(upper)
plot(lower)
r=3*sqrt(var/(n/5))
ylim([p-r,p+r])
grid()
```



The standard deviation is the square-root of the variance. As we see the in graph below, the standard deviation is 0 at the extremes: p=0,1 and is maximal at p=1/2.

In [85]:
```
P=[(i+0.0)/100.0 for i in range(101)]
std=[sqrt(P[i]*(1-P[i])) for i in range(len(P))]
plot(P,std)
```

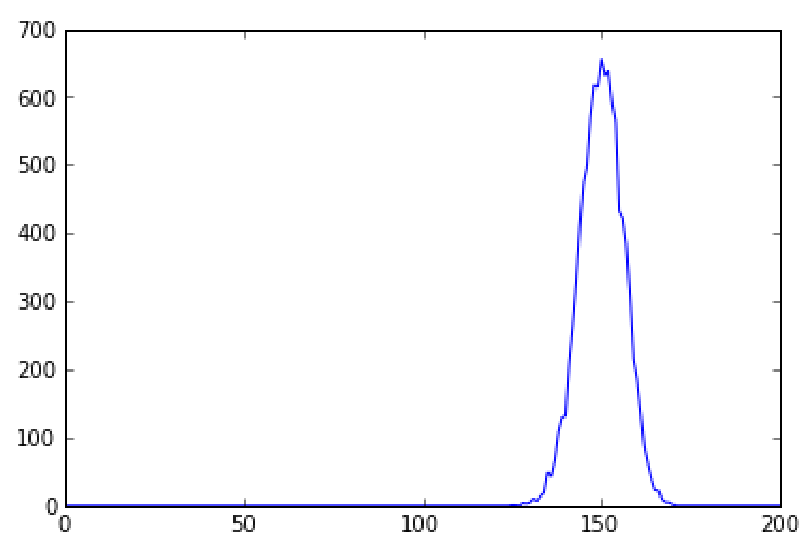
Out[85]:
```
[<matplotlib.lines.Line2D at 0x111f75510>]
```



We now plot the distribution of the sum out the outcomes for a specific sequence length *n*

In [86]:
```
n=200
m=10000
p=0.75

counts=[0]*(n+1)
for j in range(m):
    outcomes=[(1 if random()<p else 0) for i in range(n)]
    S=sum(outcomes)
    counts[S] += 1

plot(counts)
```

Out[86]:
```
[<matplotlib.lines.Line2D at 0x10ffbfa90>]
```



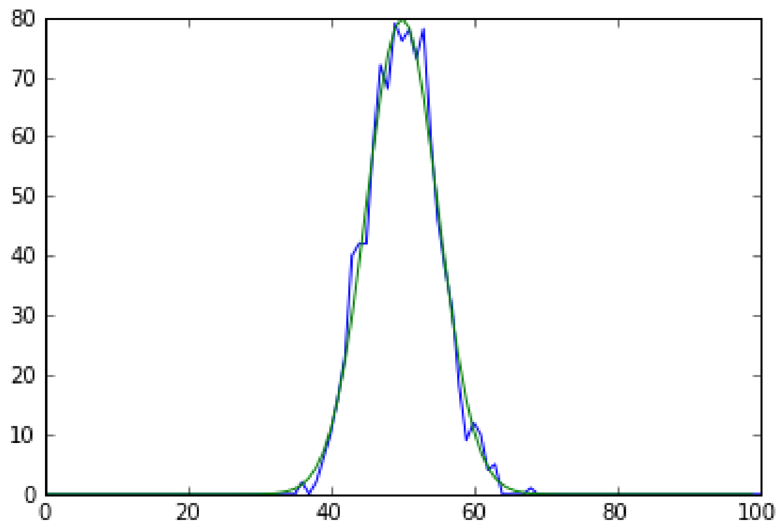
Central Limit Theorem

The central limit theorem says that the shape of the histogram above, when *n* and *m* are large, is the normal distribution, also known as the Bell curve.

In [99]:
```
n=100
m=1000
p=0.5

sigma=sqrt(n*p*(1-p))
Z=sigma*sqrt(2*pi)

counts=[0]*(n+1)
for j in range(m):
    outcomes=[(1 if random()<p else 0) for i in range(n)]
    S=sum(outcomes)
    counts[S] += 1
```

```
def nDist(i):
    diff=i-p*n
    return (m/Z)*exp(-(diff/sigma)**2/2)

ND = [nDist(i) for i in range(n) ]
plot(counts)
```

Out[99]: [<matplotlib.lines.Line2D at 0x110b60710>]



In [87]:

In [77]:

In [63]:

In [63]:

In [ ]: