Simulations using Coins

Coin Flip

We begin by implementing a coin flip. As usual, we take Head as 1 and Tail as 0.

```
def coin():
    return randint(0,1)
```

Frequency Interpretation

One interpretation of probability is that if we repeat the event, then the probability is the frequency of occurrence.

Let us check!

```
heads=0
for _ in range(10000):
    heads += coin()
N(heads/10000)
```

```
    0.502000000000000
```

Close enough!

Counting Heads

Next, we have the probability that the number of Heads is r in k tosses is given by

$$\frac{\binom{k}{r}}{2^k}$$

```
def pheads(r,k):
    return binomial(k,r)/2^k
```

We can do the frequency check for this but we need to fix k and r. Let $k = 5$ and $r = 1$.

```
num=0
for _ in range(10000):
    heads=0
    for _ in range(5):
        heads+=coin()
    if heads==1:
        num+=1
N(num/10000), N(pheads(1,5))
```

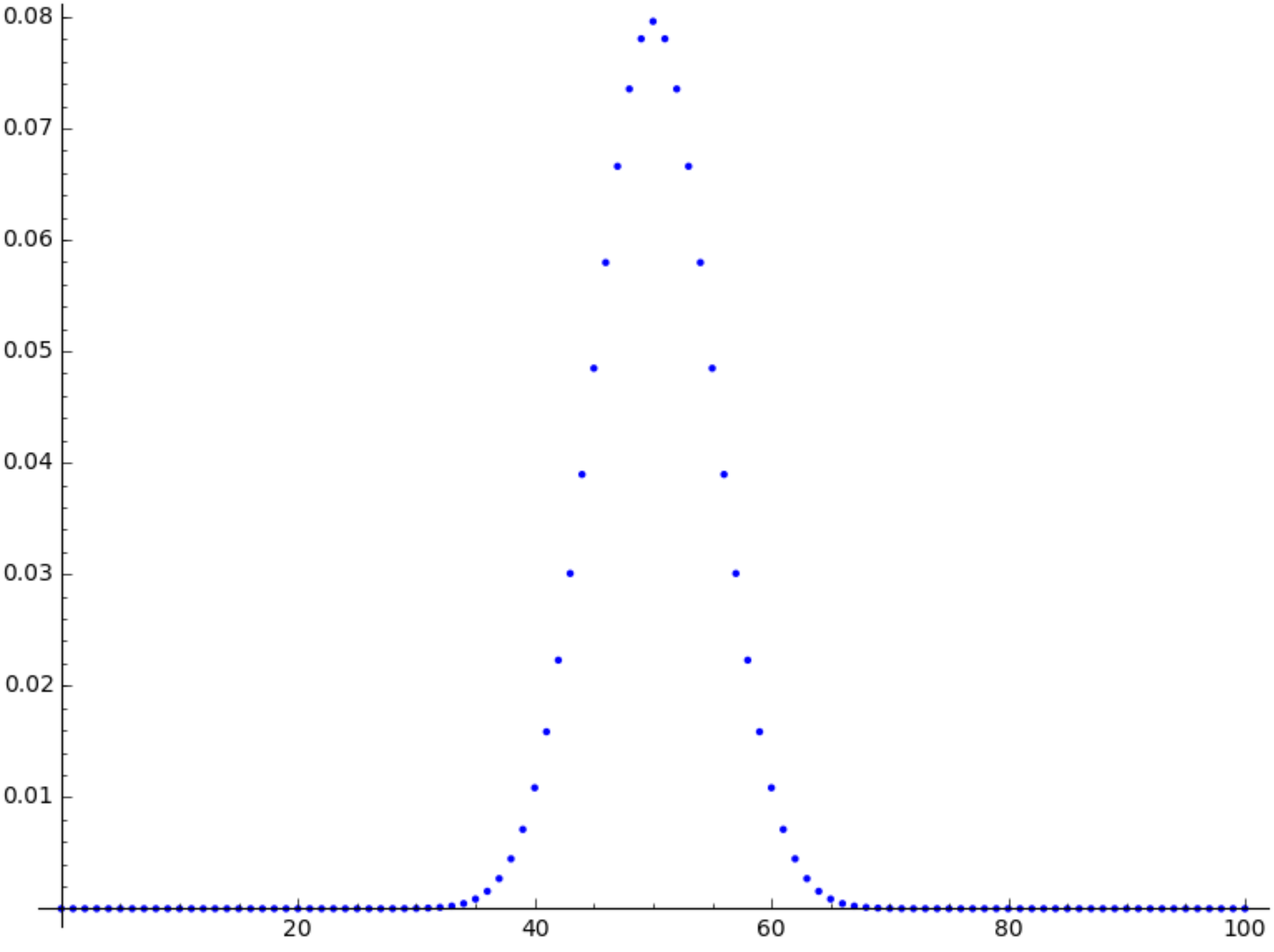
```
(0.161200000000000, 0.156250000000000)
```

Close enough!

Plotting

Let us also plot the Binomial distribution for large k to see how it looks!

```
values=[(r,N(pheads(r,100))) for r in range(101)]
list_plot(values)
```


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
20
40
60
80
100

We note that all the probabilities are low! In fact the highest value is 0.08.

Still this high probability is for 50 Heads as we expect! (Why?)

Moreover, the probability for a very small number of heads and a very small number of talks is very small. We can also expect this! (Why?)

Walks

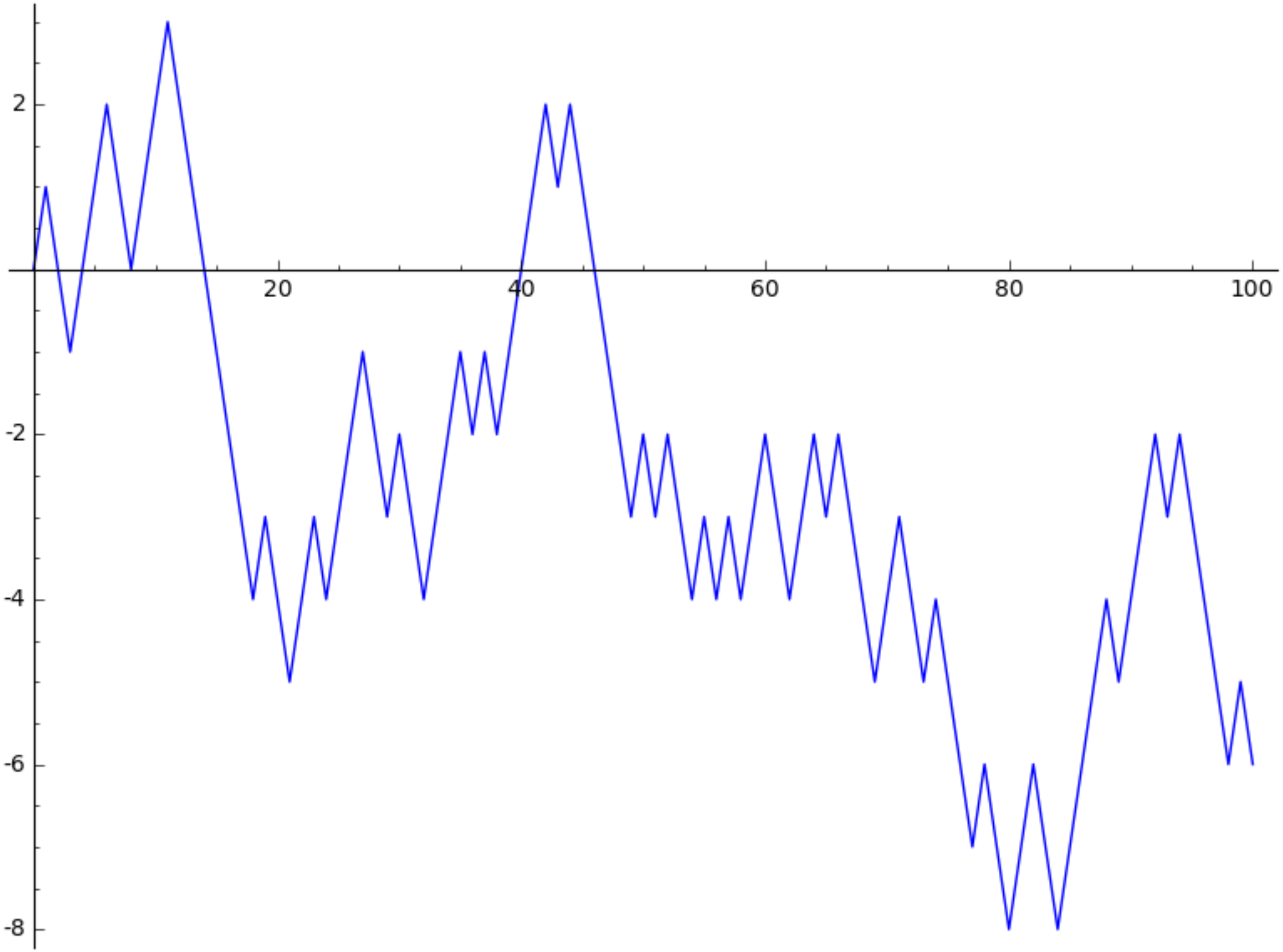
We can also simulate the "walk". One step left if Tail, or one step right if Head.

How far away are we after 100 steps?

```
def walk(n):
    steps=[2*coin()-1 for _ in range(n)]
    dist=[(i, sum(steps[:i])) for i in range(n+1)]
    return dist
```

A 100 step walk. The x axis is the time axis.

```
newwalk=walk(100)
list_plot(newwalk,plotjoined=True)
```


2
-2
-4
-6
-8
20
40
60
80
100

We see that the walk can take us quite far from the starting point!

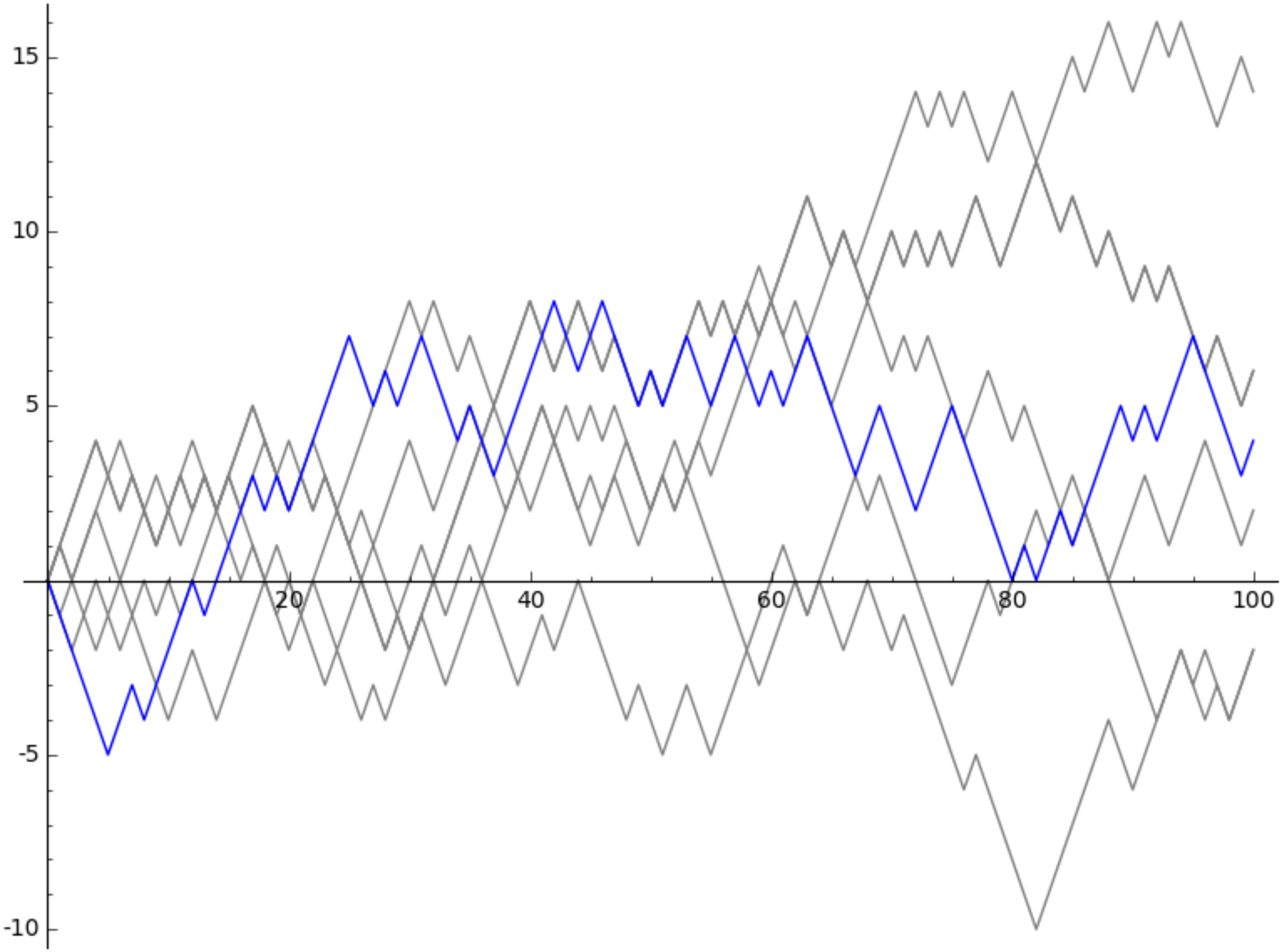
Visualising many walks

Let us save this walk and generate more and more walks.

```
p=list_plot(newwalk,plotjoined=True,color="gray")
```

Each execution of the box below plots a new walk.

```
oldwalk=newwalk
newwalk=walk(100)
p+=list_plot(oldwalk,plotjoined=True,color="gray")
p+list_plot(newwalk,plotjoined=True)
```


15
10
5
-5
-10
20
40
60
80
100

Decreasing Step Size Walk

A different walk is one where we scale the step size with each step.

```
def swalk(n):
    distance=0.0
    stepsize=1.0
    for _ in range(n):
        stepsize/=2
        distance+=(2*coin()-1)*stepsize
    return distance
```

This way we can never get below -1 or above 1!

Let us check the frequency with which we find ourseleves in a a certain subinterval of $[-1,1]$.

```
def check(a,b):
    assert (a>-1) and (b<1), "[a,b] must be a sub-interval of [-1,1]"
    num=0
    for _ in range(5000):
         d=swalk(50)
         if a<d and d<b:
             num+=1
    return N(num/5000)
```

```
for _ in range(10):
    check(0.2,0.4)
```

```
0.0938000000000000
0.104200000000000
```

```
0.101600000000000
0.101000000000000
0.0936000000000000
0.0986000000000000
0.0992000000000000
0.102800000000000
0.100600000000000
0.103600000000000
```

We see that the value is generally close to 0.1 which is half the length of the interval.

This is similar to the uniform distribution on $[-1, 1]$.

Simulation Dice

We can simulate a 6 sided die with 3 coins by declaring $1 = (0, 0, 1)$, $2 = (0, 1, 0)$, $3 = (1, 0, 0)$, $4 = (1, 1, 0)$, $5 = (1, 0, 1)$, $6 = (0, 1, 1)$.

Moreover, if we get $(0, 0, 0)$ or $(1, 1, 1)$ we throw again.

```
diedict={(0,0,1):1, (0,1,0):2, (1,0,0):3, (1,1,0):4, (1,0,1):5, (0,1,1):6}
def simdie():
    throw=(coin(),coin(),coin())
    if throw==(1,1,1) or throw==(0,0,0):
        return simdie()
    else:
        return diedict[throw]
```

We now do a frequency count to check that we are actually getting the frequency around $1/6$.

```
diecounts=[0 for _ in range(7)]
for _ in range(18000):
    diecounts[simdie()]+=1
diecounts[1:]
```

```
[3027, 3065, 2927, 2945, 3047, 2989]
```