Random Numbers

In [1]:
```
using Random
```

In [3]:
```
rand() #between 0 and 1, uniform distribtuion
```

Out[3]:
```
0.5990030773018431
```

In [4]:
```
rand(3)
```

Out[4]:
```
3-element Vector{Float64}:
 0.35051292551280433
 0.2602446878314757
 0.7724596281566194
```

```
In general you want a random number between a and b
```

In [10]:
```
function randab( a,b,n )
    return a .+ rand(n).*(b-a)
end
```

Out[10]:
```
randab (generic function with 2 methods)
```

In [11]:
```
randab( -1,1,3 )
```

Out[11]:
```
3-element Vector{Float64}:
 -0.40634028449134885
 -0.9183785263709092
  0.7627036664757694
```

How to get from another distribution, e.g. normal?

In [12]:
```
randn() #mean is 0, and std is 1
```

Out[12]:
```
0.16977740315333018
```

Plotting Histograms

Example: Let's just sample the uniform distribution

In [18]:
```
function count_histogram( num_outcomes, sequence )

    bins = zeros(num_outcomes)
    for x in sequence
        bins[x] += 1
    end
    return bins

end
```

Out[18]:
```
count_histogram (generic function with 1 method)
```

In [19]:
```
seq1 = [1,1,2,3,2,1]
```

Out[19]:
```
6-element Vector{Int64}:
 1
 1
 2
 3
 2
 1
```

In [20]:
```
count_histogram( 3,seq1 )
```

Out[20]:
```
3-element Vector{Float64}:
 3.0
 2.0
 1.0
```

LEt's say now I have 2 bins $[0, 0.5]$ and $[0.5, 1]$ if I give you 0.6, you need to tell me I'm in bin 2.

In [21]:
```
function count_unit( x,n )

    return Int( ceil(x*n) )

end
```

Out[21]:
```
count_unit (generic function with 1 method)
```

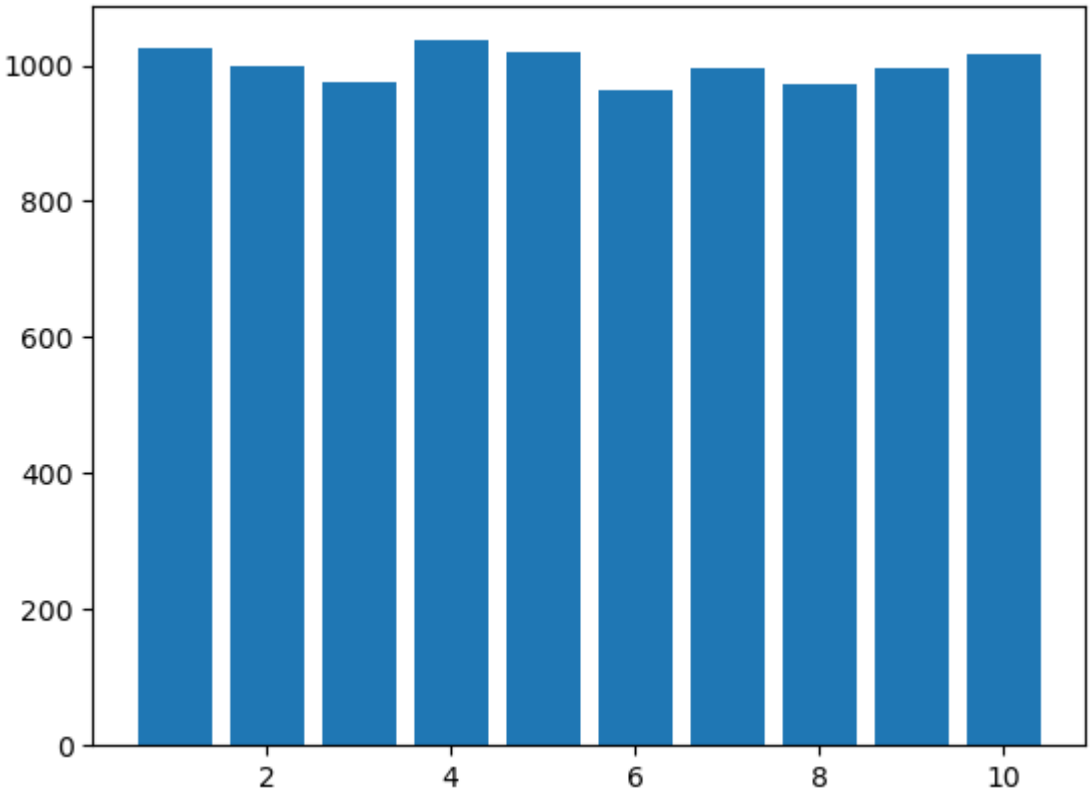
In [23]:
```
count_unit( 0.34,10 )
```

Out[23]:
```
4
```

In [ ]:
```
n = 10000
samples = rand(n);
sample_bins = [ count_unit(x,10) for x in samples ];
histogram = count_histogram( 10, sample_bins )
```

In [32]:
```
using PyPlot
```

In [33]:
```
bar( 1:10, histogram );
```



Monte Carlo Simulations

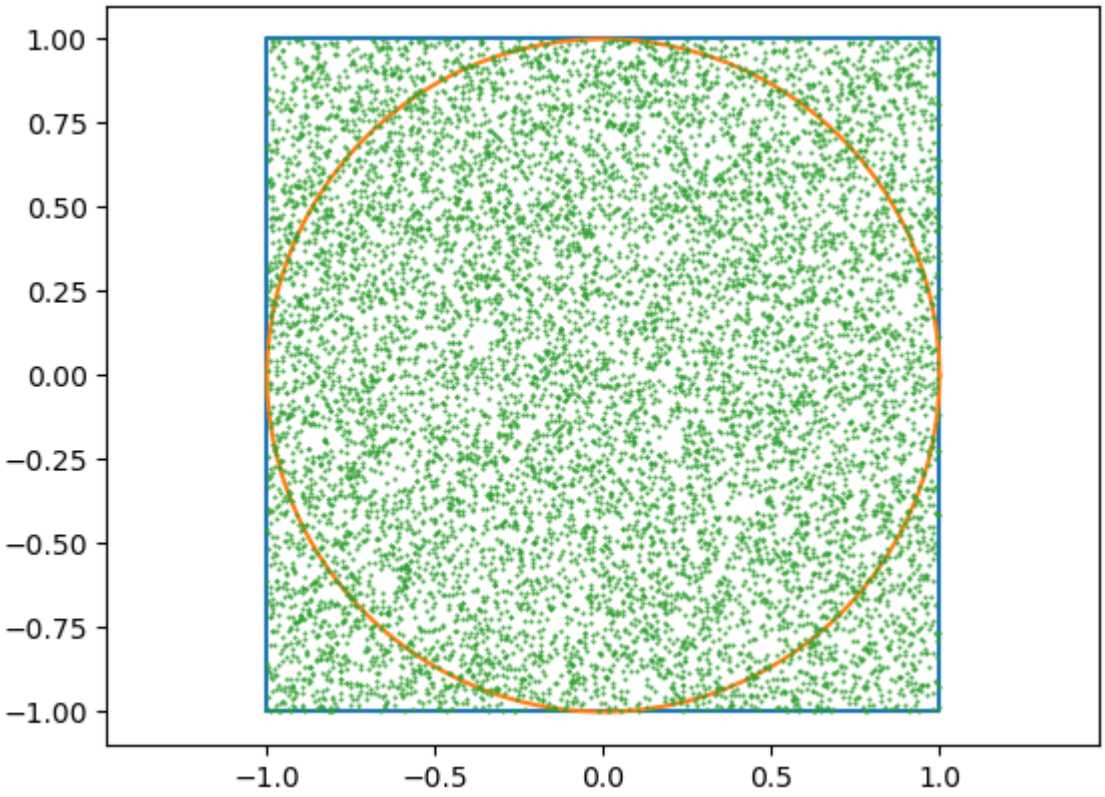
A catch all phrase for random number simulation on computers. Examples include Markov Chain MC, Kinematic MC, most famous exmaple is Metropolis MC.

Let's try to approximate π by sampling random points inside the square $[-1, 1]^2$. In theory, the probability a random point is inside the circle should be the area of the unit circle / area of the square = $\frac{\pi}{4}$

In [40]:
```
plot( [-1,1,1,-1,-1], [-1,-1,1,1,-1] )
theta = 2*pi*(0:100)./100
plot( cos.(theta), sin.(theta) )
axis("equal")

n = 10000
xvals = randab( -1,1,n )
yvals = randab( -1,1,n )

plot( xvals,yvals, linestyle="None", marker=".", markersize=1 )
```



Out[40]:
```
1-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x7f8948b1f550>
```

In [41]:
```
num_inside = 0
for i = 1:n

    x = xvals[i]; y = yvals[i];
    if x^2 + y^2 <= 1
        num_inside += 1
    end

end
```

In [42]:
```
num_inside / n * 4
```

Out[42]:
```
3.1488
```

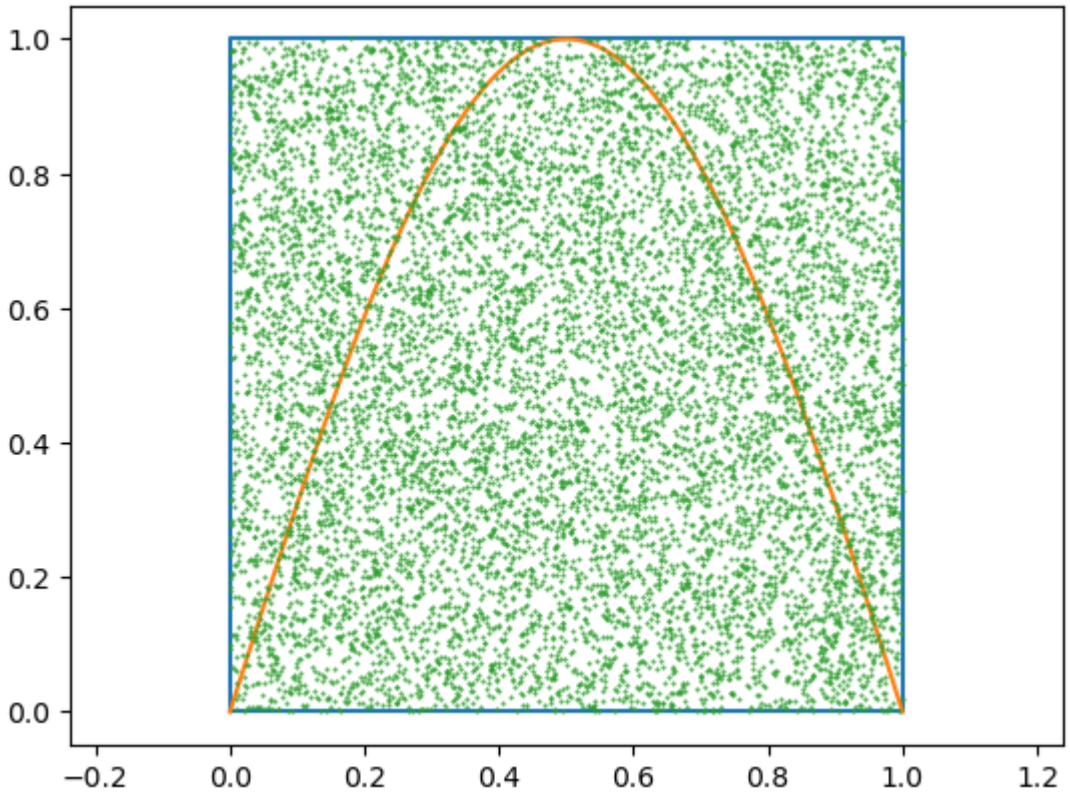
Let's now try to integrate the function $\sin(\pi x)$ between 0 and 1. This may seem to be a bit silly but is actually done a lot in practice! For example in quantum mechanics you may have a function that is defined over 100s of dimensions, and in that case standard integration techniques may be far too slow.

To do this, we can look at the number of points under the graph to approximate that area:

In [44]:
```
plot( [0,1,1,0,0], [0,0,1,1,0] )
theta = (0:100)./100
plot( (theta), sin.(pi*theta) )
axis("equal")

n = 10000
xvals = randab( 0,1,n )
yvals = randab( 0,1,n )

plot( xvals,yvals, linestyle="None", marker=".", markersize=1 )
```



Out[44]:
```
1-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x7f8948afbca0>
```

In [45]:
```
num_inside = 0
for i = 1:n

    x = xvals[i]; y = yvals[i];
    if y < sin( pi* x )
        num_inside += 1
    end

end
```

In [46]:
```
num_inside/n
```

Out[46]:
```
0.6418
```

The real answer is $2/\pi = 0.636...$

Exercise: Let's try to estimate how likely it is to win the lottery where you pick two distinct integers between 0-99. Hopefully we will see why you should never play the lottery.

To generate distinct random integers, can use the randperm function, as rand() does not guarantee uniqueness.

In [53]:
```
randperm(99)[1:2] #This was 1-99
```

Out[53]:
```
2-element Vector{Int64}:
 17
 12
```

In [54]:
```
(randperm(100).-1)[1:2]
```

Out[54]:
```
2-element Vector{Int64}:
 97
 16
```

In [56]:
```
#Can just check whether my two guesses are 0 and 1, if so I've won
n = 10000
attempts = [ (randperm(100).-1)[1:2] for i = 1:n ];
```

In [57]:
```
#Now check how many are equal to (0,1)
num_wins = 0
for i = 1:n
   attempt = attempts[i]
    if minimum(attempt) == 0 && maximum(attempt) == 1
        num_wins += 1
    end
end
```

In [58]:
```
num_wins
```

Out[58]:
```
3
```

In [59]:
```
num_wins/n
```

Out[59]:
```
0.0003
```

In [64]:
```
abs(rand(Int)%100)
```

Out[64]:
```
71
```

In [66]:
```
(-1%4)+4
```

Out[66]:
```
3
```

In [67]:
```
mod(-1,4)
```

Out[67]:
```
3
```

In [ ]: