Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2023

Homework 4

Due Wednesday, February 15

Problem 1

(from **Insight**, P6.1.10)

Write a program that generates $n = 10$ real numbers selected randomly and uniformly distributed from the set $\{x : 0 < x < 2 \text{ or } 7 < x < 10\}$.

In [22]:
```
function random()
a = []
    while length(a) < 10
        b = 10*rand()
        if 0 < b < 2 || 7 < b < 10
            push!(a, b)
        else
        end
    end
a
end
```

Out[22]:
```
random (generic function with 1 method)
```

In [48]:
```
random()
```

Out[48]:
```
10-element Vector{Any}:
 1.1551650161894922
 1.027144092933241
 8.416292551153573
 7.059121078662431
 9.164274683371332
 9.458481630874944
 8.120783848495583
 8.912327696266662
 1.1542180920898337
 8.748218340421614
```

Problem 2

(from **Insight**, P6.1.12)

Assume that the coefficients of the quadratic $ax^2 + bx + c$ are selected from the uniform distribution on $(-2, 2)$. Use the Monte Carlo method to determine the probability of complex roots. What if the coefficients are generated with `randn` with mean $\mu = 0$ and standard deviation $\sigma = 0.4$?

In [89]:
```
function complex_quad_prob1(n)
    ntrials = 0
    ncomplex = 0
    for i = 1:n
        a = 4rand(3) .- 2
        if a[2]^2 - 4a[1]a[3] < 0
            ncomplex += 1
        else
        end
        ntrials += 1
    end
    println("Probability of complex roots: " , ncomplex / n)
end
```

Out[89]:
```
complex_quad_prob1 (generic function with 1 method)
```

In [90]:
```
complex_quad_prob1(20000)
```

```
Probability of complex roots: 0.37695
```

In [91]:
```
function complex_quad_prob2(n)
    ntrials = 0
    ncomplex = 0
    for i = 1:n
        a = .4randn(3)
        if a[2]^2 - 4a[1]a[3] < 0
            ncomplex += 1
        else
        end
        ntrials += 1
    end
    println("Probability of complex roots: " , ncomplex / n)
end
```

Out[91]:
```
complex_quad_prob2 (generic function with 1 method)
```

In [96]:
```
complex_quad_prob2(20000)
```

```
Probability of complex roots: 0.3528
```

Problem 3

(from **Insight**, P6.1.15)

Two points on the unit circle are randomly selected. Use the Monte Carlo method to determine the probability that the length of the connecting chord is greater than 1.

In [107]:
```
function chord_length_prob(n)
    nsuccess = 0
    for i = 1:n
        a = 2π*rand(2)
        p1 = [cos(a[1]), sin(a[1])]
        p2 = [cos(a[2]), sin(a[2])]
        chord = (p1 .- p2)
        if sqrt(chord[1]^2 + chord[2]^2) > 1
            nsuccess += 1
        else
        end
    end
    println("Probablity of chord length > 1: " , nsuccess / n)
end
```

Out[107]:
```
chord_length_prob (generic function with 1 method)
```

In [112]:
```
chord_length_prob(20000)
```

```
Probablity of chord length > 1: 0.6621
```

Problem 4

(from **Insight**, P6.1.19a)

Write a function `ProbG(L,R)` that returns an estimate of the area under the function

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

from L to R using Monte Carlo. Assume $L < R$. *Hint*: Throw darts in the rectangle having vertices $(L, 0)$, $(R, 0)$, $(R, 1)$, and $(L, 1)$ and count how many are under the curve.

In [47]:
```
function ProbG(L,R)
    ntrials = 100000
    x = (R - L)rand(ntrials) .+ L
    y = rand(ntrials)
    P = [x y]
    hit = 0
    for i = 1:size(P,1)
        if 1/(sqrt(2π))*exp((-P[i,1]^2)/2) ≥ P[i,2]
            hit += 1
        else
        end
    end
    println("Area under f(x) = ", hit/ntrials*(R - L))
end
```

Out[47]:
```
ProbG (generic function with 1 method)
```

Test the function using $L = 0$ and $R = 2$:

In [49]:
```
ProbG(0,2)
```

```
Area under f(x) = 0.47854
```

Problem 5

(from **Insight**, P6.2.5)

Consider the random walk function `random_walk(n)` from the lecture slides.

We conjecture that the walker is more likely to exit near the middle of an edge than near a corner. Produce a bar plot that sheds light on this conjecture, for `n=20` and a large number of trials.

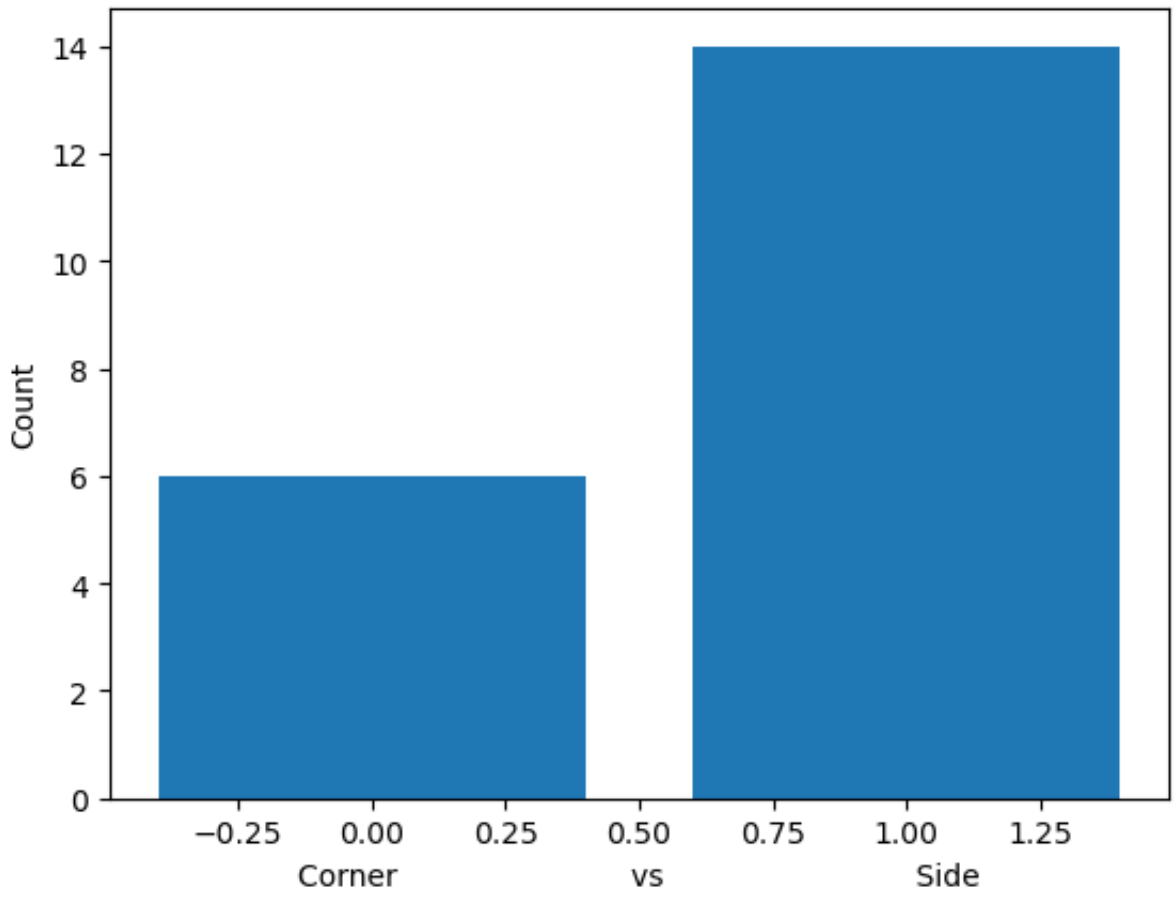
In [70]:
```
function random_walk(n)
    x = [0]
    y = [0]
    while abs(x[end]) < n && abs(y[end]) < n
        if rand() < .5
            if rand() < .5
                push!(x, x[end]+1)
                push!(y, y[end])
            else
                push!(x, x[end]-1)
                push!(y, y[end])
            end
        else
            if rand() < .5
                push!(y, y[end]+1)
                push!(x, x[end])
            else
                push!(y, y[end]-1)
                push!(x, x[end])
            end
        end
    end
    p = [x[end] y[end];]
end

```

Out[70]: `random_walk (generic function with 1 method)`

In [59]:
```
using PyPlot
```

In [133]:
```

```

```
random_walk(20)

function walk_hist(n, m)
    corner = 0
    side = 0
    for i = 1:m
        p = random_walk(n)
        if abs(p[1]) < n/2
            side += 1
        else
            if abs(p[2]) < n/2
                side += 1
            else
                corner += 1
            end
        end
    end
    bar([0,1], Int64[corner , side])
    xlabel("Corner                          vs
    ylabel("Count")

end

walk_hist(1000,20)
```



Out[133]: PyObject Text(24.000000000000007, 0.5, 'Count')

Problem 6

Use Monte Carlo simulation with one million trials to estimate the probability that a random poker hand contains two pairs (that is, two of each of two different ranks and a fifth card of a third rank).

In [20]:

```
function prob_2pair(n)
    r = collect(1:52)
    for i = 1:52
        r[i] = (i - 1) % 13 + 1
    end

    s = collect(1:52)
    for i = 1:52
        s[i] = (i - 1) ÷ 13 + 1
    end

    D = [r s]
    count = 0
    for i = 1:n
        H = [0 0]
        row = zeros(Int64,5)
        for i = 1:5
            works = 0
            while works != 4
                works = 0
                row[i] = rand(1:52)
                for j = 1:5
                    if row[i] != row[j]
                        works += 1
                    else
                    end
                end
            end
            row
        end

        for j = 1:5
            H = [H;[D[row[j], :][1] D[row[j], :][2]]]
        end

        S = []
        for j = 1:size(H, 1) - 1
            for i = j:size(H, 1) - 1
                if H[i,1] == H[i+1,1]
```

```
                    push!(S, H[j,1])
                    push!(S, H[i+1,1])
                else
                end
            end
        end
        if length(S) == 4
            if S[1] == S[2] == S[3] == S[4]
                break
            else
            count += 1
            end
        else
        end
    end
    println("Probability of 2 pairs = ", count/n)
end
```

Out[20]: prob_2pair (generic function with 1 method)

In [22]:
```
prob_2pair(100000)
```

```
Probability of 2 pairs = 0.04741
```

Problem 7

Use array functions and vectorization to solve the problems below using *only a single line of code* for each problem.

In [24]:
```
A = reshape((-22:22) .% 11, 9, 5)          # For testing
```

Out[24]:
```
9×5 Matrix{Int64}:
   0   -2  -4   5   3
 -10   -1  -3   6   4
  -9    0  -2   7   5
  -8  -10  -1   8   6
  -7   -9   0   9   7
  -6   -8   1  10   8
  -5   -7   2   0   9
  -4   -6   3   1  10
  -3   -5   4   2   0
```

Problem 7(a)

Count the number of elements a of A that satisfy $a^2 < 10$.

In [46]:
```
gen = sum((1 for a in A if abs(a) < sqrt(10)))
```

Out[46]: 17

Problem 7(b)

Create a matrix which contains only the columns j of A where the first element $A_{1,j} \geq 0$.

In [74]:
```
B=[ for j = 1:size(A,2) if A[1,j] ≥ 0]
```

```
MethodError: no method matching reshape(::Int64, ::Int64)
Closest candidates are:
  reshape(::AbstractArray, ::Int64...) at reshapedarray.jl:116
  reshape(::AbstractArray, ::Union{Int64, AbstractUnitRange}...) at r
eshapedarray.jl:110
  reshape(::AbstractArray, ::Union{Colon, Int64}...) at reshapedarray
.jl:117

Stacktrace:
  [1] (::var"#147#151"{Int64})(i::Int64)
    @ Main ./none:0
  [2] iterate
    @ ./generator.jl:47 [inlined]
  [3] iterate
    @ ./iterators.jl:1095 [inlined]
  [4] iterate
    @ ./iterators.jl:1089 [inlined]
  [5] grow_to!(dest::Vector{Any}, itr::Base.Iterators.Flatten{Base.Ge
nerator{Base.Iterators.Filter{var"#149#152", UnitRange{Int64}}, var"#
148#150"}})
    @ Base ./array.jl:743
  [6] _collect
    @ ./array.jl:652 [inlined]
  [7] collect(itr::Base.Iterators.Flatten{Base.Generator{Base.Iterato
rs.Filter{var"#149#152", UnitRange{Int64}}, var"#148#150"}})
    @ Base ./array.jl:602
  [8] top-level scope
    @ In[74]:1
  [9] eval
    @ ./boot.jl:360 [inlined]
 [10] include_string(mapexpr::typeof(REPL.softscope), mod::Module, co
de::String, filename::String)
    @ Base ./loading.jl:1116
```

Problem 7(c)

Modify A in the following way: Multiply all the elements that are even by 3 (you might need to print A on a separate line to see the full matrix):

In [ ]:
```
1
```