Assignment instructions: Completing this assignment requires submitting, for each problem, a written part or a `Matlab` program, and in some cases, both of these. Any written part should be handed in during class; Matlab programs should be emailed to `esam448@u.northwestern.edu`. Please comment your code to explain your implementation. Including *all* of the programs for a given assignment in a single email will be appreciated.

1. A derangement of the numbers 1 through N is a permutation of all N of those numbers such that none of them is in the "right place." For example, 34251 is a derangement of 1 through 5, but 24351 is not because 3 is in the 3rd position. Write a Matlab function (using the specific function name `lastname_firstname_hw1_prob1`)

```
function [prob_est,stderr_of_est] = lastname_firstname_hw1_prob1(M,N)

[...any statements you need...]

end
```

that generates M random permutations of the integers $1, 2, \ldots, N$ and then determines how many of those permutations is a derangement. The function should return an estimate for the probability of a derangement occurring, i.e., $P\{$ a permutation of $1,\ldots,N$ is a derangement$\}$ *and* an estimate for the standard error associated with that estimate.

Use your Matlab function to determine the estimated probability when $N = 12$ and report the number of samples necessary to reduce the standard error in that estimate to 0.0001.

Solution:

```
function [prob_est,stderr_of_est] = hw_solution_hw1_prob1(M,N)

ntrial=M;
[~, A] = sort(rand(ntrial,N),2);     % index from sort gives permutations
B=repmat(1:N,ntrial,1);              % make ntrial vectors 1:N
zeroflag=sum((A-B)==0,2)==0;         % check if any digits in same position
prob_est=mean(zeroflag);             % estimate is mean
sig2=var(zeroflag);                  % sigma^2 is variance
stderr_of_est=sqrt(sig2/ntrial);     % standard error of the mean

end
```

We run this with a script and get the output:

```
>> run_hw1_prob1
probability of derangement is   0.3679 with sem   0.000482
number of samples for stderr to be   0.0001 is 23255246
```

Here's the script:

```
M=1000000;
[prob,sem] = hw_solution_hw1_prob1(M,12);
fprintf(1,'probability of derangement is %8.4f with sem %10.6f\n',prob,sem);
target=0.0001;
fprintf(1,'number of samples for stderr to be %8.4f is %i\n',...
    target,floor(M*(sem/target)^2+0.5));
```

2. Consider the following game: you begin with \$K. You flip a coin — that may or may not be fair — winning \$1 if the coin lands heads and losing \$1 if the coin lands tails. The coin flips continue until you either go broke or have \$100. Write a Matlab function that takes the starting value `K`, a number of samples `M`, and a probability `p` of heads

```
function [prob_est,stderr_of_est] = lastname_firstname_hw1_prob2(M,K,p)

[...any statements you need...]

end
```

that estimates the probability of reaching \$100 and the estimate's standard error from `M` samples. (Here, one sample is considered to be one sequence of coin flips that reaches either \$100 or \$0.)

First, check your program for the case `p=0.5` and K=50; by symmetry the probability of reaching \$100 should be 0.5 in this case. Then determine the probability of reaching \$100 if the probability of the coin producing heads on a single flip is reduced to `p=0.49`.

Solution: Here is a solution. This version also computes the average *time* it takes to win or lose.

```
function [prob_est,stderr_of_est] = hw_solution_hw1_prob2(M,K,p)

state=floor(K/2+0.5)*ones(M,1);   % state vector; start halfway
count=zeros(M,1);                 % number of times played
nactive=M;                        % number still playing

while (nactive>0)                 % stop if everyone done
    winlose=2*(rand(nactive,1)<p)-1;    % +1 if win, -1 if lose
    count(1:nactive)=count(1:nactive)+1;   % increment times played
    state(1:nactive)=state(1:nactive)+winlose;  % change state if active
    nactive=sum((state<K)&(state>0));      % update number active
    % re-sort state and count vectors; active players first
    state=[state((state<K)&(state>0)); state(state==K); state(state==0)];
    count=[count((state<K)&(state>0)); count(state==K); count(state==0)];
```

```
end

wincounts=count(state==K);  % number of times played to win
losecounts=count(state==0); % number of times played to lose

wincountmean=mean(wincounts);           % mean time to win
losecountmean=mean(losecounts);             % mean time to loss
wincountsem=sqrt(var(wincounts)/length(wincounts)); % sem of win time
losecountsem=sqrt(var(losecounts)/length(losecounts));   % sem - lose time

state=state/K;          % normalize win/lose vector
prob_est=mean(state);   % mean is prob of winning
winvar=var(state);     % estimate of variance
stderr_of_est=sqrt(winvar/M);    % std error of estimate
end
```

Running this script

```
[prob_est,stderr_of_est]=hw_solution_hw1_prob2(100000,100,0.5);
fprintf(1,'probability of winning is %8.4f with sem %10.6f\n',prob_est,stderr_of_est);
```

gives

```
probability of winning is   0.4996 with sem   0.001581
```

and changing p to 0.49 gives instead

```
probability of winning is   0.1176 with sem   0.001019
```

Note that a tiny shift in the overall win probability makes a big difference on the probability of hitting $100.

3. Consider picking three points in the (x, y) plane where the x and y coordinates of each point are independently chosen from a standard normal distribution. These three points, of course, determine a triangle. Write a Matlab function

```
function [prob_est,stderr_of_est] = lastname_firstname_hw1_prob3(M)

[...any statements you need...]

end
```

that uses M independently sampled triangles to determine the probability that such a randomly chosen triangle is obtuse. Also return the standard error of the estimate.

User your Matlab function to estimate this probability. Can you make a guess as to what the right answer should be? Do you think it's reasonable to try to justify this guess by increasing the number of samples?

Solution: The most difficult part here is figuring out how to test if the triangle is obtuse; one way is via the law of cosines. To apply this easily, in vector format, we just compute the lengths of the sides and then sort those lengths so that the longest side is last.

```
function [prob_est,stderr_of_est] = hw_solution_hw1_prob3(M)

X=randn(M,3);        % 3 X coordinates
Y=randn(M,3);        % 3 Y coordinates
D=zeros(M,3);        % lengths of sides

% distance of first, second and third sides
D(:,1)=sqrt((X(:,2)-X(:,1)).^2+(Y(:,2)-Y(:,1)).^2);
D(:,2)=sqrt((X(:,3)-X(:,2)).^2+(Y(:,3)-Y(:,2)).^2);
D(:,3)=sqrt((X(:,1)-X(:,3)).^2+(Y(:,1)-Y(:,3)).^2);

D=sort(D,2);    % sort in inreasing order
obtuse=(D(:,1).^2+D(:,2).^2<D(:,3).^2);   % law of cosines (!)

prob_est=mean(obtuse);
stderr_of_est=sqrt(var(obtuse)/M);

end
```

Here's a script

```
[prob,sem]=hw_solution_hw1_prob3(1000000);
fprintf(1,'fraction obtuse is %10.4f; sem = %10.4f\n',prob,sem);
```

and the output

```
>> run_hw1_prob3
fraction obtuse is     0.7501; sem =     0.0004
```

It certainly looks like the exact probability is 3/4; it's not too hard to increase the number of samples to get one or two more zeros. While this cannot be considered a proof of the result, it is certainly suggestive that there should be some way to show it.

4. Use a change of variable with an appropriate transformation and Matlab's uniform `rand` function to generate random values from an exponential probability distribution, $p(x) = ae^{-ax}$, $x \geq 0$. Your write-up should give the details explaining how this works. Then for $a = 1$ generate $N = 1000$ samples and make a plot [include this in your written solutions] comparing the sample cumulative distribution function (`Matlab` function `ecdf`) with the exact answer. Next, write a Matlab function with the structure

```
function [chi2pvals] = lastname_firstname_hw1_prob4(M,N,a,nbins)
chi2pvals=zeros(M,1)

[...other statements you need...]

end
```

This function should generate a vector of N exponential random variables with parameter a multiple (M) times. For each N-vector of samples, it should use the χ^2 test (`Matlab`: `chi2gof`) to compare the generated data with the theoretical distribution. For the χ^2 test, divide the full range from 0 to ∞ into `nbins` in such a way so that the expected number of samples falling in each bin will be equal (note: these will not be equal width bins). The function should return a vector giving the M p-values from the χ^2 tests. If `M=1000`, `N=1000`, `a=1` and `nbins=10`, how many times does the vector of samples pass the χ^2 test at a 0.95 significance? For these parameters, include in your written solution a plot showing a histogram of the p-values using 20 equal width bins. What does the distribution of p-values look like? Can you explain this? Finally, include your function in the email to `esam448@u.northwestern.edu`.

Solution: To generate these samples we need the cumulative distribution function (cdf)

$$F(x) = \int_0^x ae^{-a\xi}\, d\xi = 1 - e^{-ax}\,.$$

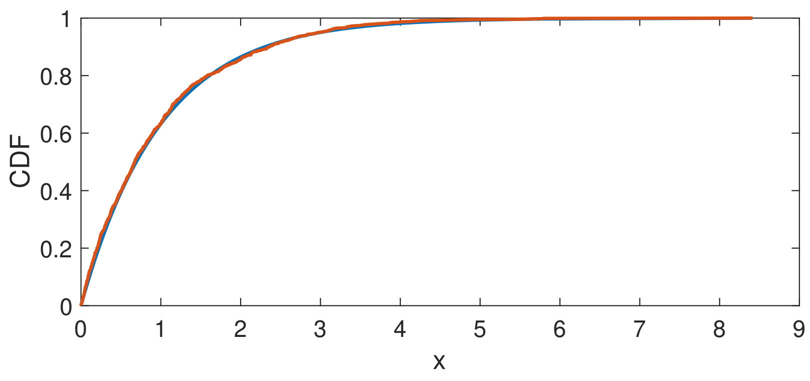
In addition, we then have the inverse.

$$F^{-1}(x) = -\frac{1}{a}\ln(1-x)\,.$$

As described in the course notes, we therefore draw uniform random variables U using Matlab's `rand` function and generate exponential random variables using $X = -\ln(1-U)/a$. Here is the code to generate the `ecdf` and to call the function to generate the set of p-values and then plot the p-value histogram:

```
a=1;         % set parameters
M=1000;
N=1000;

X=-log(1-rand(N,1))/a;      % generate exponential samples

figure(1);
[f,y]=ecdf(X);              % numerical CDF plot
plot(y,1-exp(-a*y),y,f,'linewidth',2);
ax=gca;
set(ax,'linewidth',1,'FontSize',14);
xlabel('x','FontSize',16);
ylabel('CDF','FontSize',16);
set(ax,'PlotBoxAspectRatio',[1.0 0.4 1.0]);

nbins=10;                   % generate vector of chi2 p-values
[chi2pvals]=hw_solution_hw1_prob4(M,N,a,nbins);

figure(2);
hist(chi2pvals,20)          % make histogram of p-values
ax=gca;
set(ax,'linewidth',1,'FontSize',14);
set(ax,'PlotBoxAspectRatio',[1.0 0.4 1.0]);
xlabel('p-value','FontSize',16);
ylabel('#','FontSize',16);
saveas(gcf,'p-val_hist.eps','epsc')
```

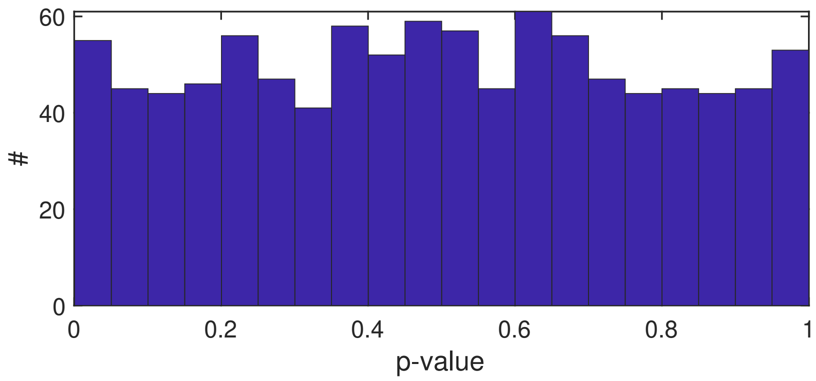
and here is the `ecdf`, including the exact CDF for comparison:



Here is the function for computing the p-values:

```
function [chi2pvals] = hw_solution_hw1_prob4(M,N,a,nbins)
chi2pvals=zeros(M,1);       % pre-allocate vector

chi2bins=-log(1-(0:nbins)/nbins)/a;     % bins for chi2 test
expt=N/nbins*ones(1,nbins);             % expected number of observations

for k=1:M
    X=-log(1-rand(N,1))/a;            % exponentially distributed samples
    obsv=histcounts(X,chi2bins);      % bin samples
    chi2val=sum((obsv-expt).^2./expt);  % chi2 statistic
    chi2pvals(k)=chi2cdf(chi2val,nbins-1,'upper');  % chi2 tail prob
end
end
```

and here is an example histogram of p-values:



Note that p-values should have a uniform distribution if the null-hypothesis is true. This is because the test has been designed so that when the null hypothesis is true the probability for the test statistic T to be larger than some prescribed value *is* the p-value — in other words, the p-value is $1 - F(T)$, where $F(T)$ is the CDF of the test statistic. By definition the value of the CDF should be uniformly distributed. Thus, in this case, where the significance level is 0.95, this means that we want the p-values to be larger than 0.95, and this should happen about 5% of the time. For 1,000 samples, this means that roughly 50 times the p-value should be smaller than 0.05.