Contents

Answers to week 3 exercises (some at least)

```
clc
close all
format short
```

Exercise set 1

```
% a)
help randi

x = randi([4 12],1,20)
```

```
 RANDI Pseudorandom integers from a uniform discrete distribution.
    R = RANDI(IMAX,N) returns an N-by-N matrix containing pseudorandom
    integer values drawn from the discrete uniform distribution on 1:IMAX.
    RANDI(IMAX,M,N) or RANDI(IMAX,[M,N]) returns an M-by-N matrix.
    RANDI(IMAX,M,N,P,...) or RANDI(IMAX,[M,N,P,...]) returns an
    M-by-N-by-P-by-... array.  RANDI(IMAX) returns a scalar.
    RANDI(IMAX,SIZE(A)) returns an array the same size as A.

    R = RANDI([IMIN,IMAX],...) returns an array containing integer
    values drawn from the discrete uniform distribution on IMIN:IMAX.

    Note: The size inputs M, N, P, ... should be nonnegative integers.
    Negative integers are treated as 0.

    R = RANDI(..., CLASSNAME) returns an array of integer values of class
    CLASSNAME.

    R = RANDI(..., 'like', Y) returns an array of integer values of the
    same class as Y.

    The arrays returned by RANDI may contain repeated integer values.  This
    is sometimes referred to as sampling with replacement.  To get unique
    integer values, sometimes referred to as sampling without replacement,
    use RANDPERM.

    The sequence of numbers produced by RANDI is determined by the settings of
    the uniform random number generator that underlies RAND, RANDN, and RANDI.
    RANDI uses one uniform random value to create each integer random value.
    Control that shared random number generator using RNG.

    Examples:

       Example 1: Generate integer values from the uniform distribution on
       the set 1:10.
          r = randi(10,100,1);

       Example 2: Generate an integer array of integer values drawn uniformly
       from 1:10.
          r = randi(10,100,1,'uint32');

       Example 3: Generate integer values drawn uniformly from -10:10.
          r = randi([-10 10],100,1);

       Example 4: Reset the random number generator used by RAND, RANDI, and
       RANDN to its default startup settings, so that RANDI produces the same
       random numbers as if you restarted MATLAB.
          rng('default');
          randi(10,1,5)

       Example 5: Save the settings for the random number generator used by
       RAND, RANDI, and RANDN, generate 5 values from RANDI, restore the
       settings, and repeat those values.
          s = rng
          i1 = randi(10,1,5)
          rng(s);
          i2 = randi(10,1,5) % i2 contains exactly the same values as i1

       Example 6: Reinitialize the random number generator used by RAND,
       RANDI, and RANDN with a seed based on the current time.  RANDI will
```

```
        return different values each time you do this.  NOTE: It is usually
        not necessary to do this more than once per MATLAB session.
           rng('shuffle');
           randi(10,1,5)

     See also RAND, RANDN, RANDPERM, RNG, RANDSTREAM

     Reference page in Doc Center
        doc randi

     Other functions named randi

        RandStream/randi


 x =

   Columns 1 through 13

     12    10     8    12     4    10    10     9     5     9     6     5     5

   Columns 14 through 20

     12     4     6     4     7     4    12

```

b)

```
help rand

x = 8*rand(1,20)-2
```

```
  RAND Uniformly distributed pseudorandom numbers.
     R = RAND(N) returns an N-by-N matrix containing pseudorandom values drawn
     from the standard uniform distribution on the open interval(0,1).  RAND(M,N)
     or RAND([M,N]) returns an M-by-N matrix.  RAND(M,N,P,...) or
     RAND([M,N,P,...]) returns an M-by-N-by-P-by-... array.  RAND returns a
     scalar.  RAND(SIZE(A)) returns an array the same size as A.

     Note: The size inputs M, N, P, ... should be nonnegative integers.
     Negative integers are treated as 0.

     R = RAND(..., CLASSNAME) returns an array of uniform values of the
     specified class. CLASSNAME can be 'double' or 'single'.

     R = RAND(..., 'like', Y) returns an array of uniform values of the
     same class as Y.

     The sequence of numbers produced by RAND is determined by the settings of
     the uniform random number generator that underlies RAND, RANDI, and RANDN.
     Control that shared random number generator using RNG.

     Examples:

        Example 1: Generate values from the uniform distribution on the
        interval (a, b).
           r = a + (b-a).*rand(100,1);

        Example 2: Use the RANDI function, instead of RAND, to generate
        integer values from the uniform distribution on the set 1:100.
           r = randi(100,1,5);

        Example 3: Reset the random number generator used by RAND, RANDI, and
        RANDN to its default startup settings, so that RAND produces the same
        random numbers as if you restarted MATLAB.
           rng('default')
           rand(1,5)

        Example 4: Save the settings for the random number generator used by
        RAND, RANDI, and RANDN, generate 5 values from RAND, restore the
        settings, and repeat those values.
           s = rng
           u1 = rand(1,5)
           rng(s);
           u2 = rand(1,5) % contains exactly the same values as u1

        Example 5: Reinitialize the random number generator used by RAND,
        RANDI, and RANDN with a seed based on the current time.  RAND will
        return different values each time you do this.  NOTE: It is usually
        not necessary to do this more than once per MATLAB session.
```

```
         rng('shuffle');
         rand(1,5)

   See <a href="x" '\techdoc\math\math.map'],'update_random_number_generator')">Replace Discouraged Syntaxes of rand and randn</a> to use RNG to replace
   RAND with the 'seed', 'state', or 'twister' inputs.

   See also RANDI, RANDN, RNG, RANDSTREAM, RANDSTREAM/RAND,
            SPRAND, SPRANDN, RANDPERM.

   Reference page in Doc Center
      doc rand

   Other functions named rand

      RandStream/rand


x =

  Columns 1 through 7

   -0.4267   -1.2530    0.4589    1.6485   -1.1866    5.9631    0.6567

  Columns 8 through 14

    0.3788   -1.5036    0.3860   -1.6292    2.0434    4.0914    3.0486

  Columns 15 through 20

   -1.2809   -1.3531    4.2179    5.2411    2.2702   -1.1268
```

first scale the interval (0,1) to the correct width then shift

c)

```
help randn

x = 3*randn(1,40)+2
```

```
 RANDN Normally distributed pseudorandom numbers.
    R = RANDN(N) returns an N-by-N matrix containing pseudorandom values drawn
    from the standard normal distribution.  RANDN(M,N) or RANDN([M,N]) returns
    an M-by-N matrix. RANDN(M,N,P,...) or RANDN([M,N,P,...]) returns an
    M-by-N-by-P-by-... array. RANDN returns a scalar.  RANDN(SIZE(A)) returns
    an array the same size as A.

    Note: The size inputs M, N, P, ... should be nonnegative integers.
    Negative integers are treated as 0.

    R = RANDN(..., CLASSNAME) returns an array of normal values of the
    specified class. CLASSNAME can be 'double' or 'single'.

    R = RANDN(..., 'like', Y) returns an array of normal values of the
    same class as Y.

    The sequence of numbers produced by RANDN is determined by the settings of
    the uniform random number generator that underlies RAND, RANDN, and RANDI.
    RANDN uses one or more uniform random values to create each normal random
    value.  Control that shared random number generator using RNG.

    Examples:

       Example 1: Generate values from a normal distribution with mean 1
        and standard deviation 2.
          r = 1 + 2.*randn(100,1);

       Example 2: Generate values from a bivariate normal distribution with
       specified mean vector and covariance matrix.
          mu = [1 2];
          Sigma = [1 .5; .5 2]; R = chol(Sigma);
          z = repmat(mu,100,1) + randn(100,2)*R;

       Example 3: Reset the random number generator used by RAND, RANDI, and
       RANDN to its default startup settings, so that RANDN produces the same
       random numbers as if you restarted MATLAB.
          rng('default');
          randn(1,5)

       Example 4: Save the settings for the random number generator used by
```

```
        RAND, RANDI, and RANDN, generate 5 values from RANDN, restore the
        settings, and repeat those values.
           s = rng
           z1 = randn(1,5)
           rng(s);
           z2 = randn(1,5) % z2 contains exactly the same values as z1

        Example 5: Reinitialize the random number generator used by RAND,
        RANDI, and RANDN with a seed based on the current time.  RANDN will
        return different values each time you do this.  NOTE: It is usually
        not necessary to do this more than once per MATLAB session.
           rng('shuffle');
           randn(1,5)

     See <a href="x" '\techdoc\math\math.map'],'update_random_number_generator')">Replace Discouraged Syntaxes of rand and randn</a> to use RNG to replace
     RANDN with the 'seed' or 'state' inputs.

     See also RAND, RANDI, RNG, RANDSTREAM, RANDSTREAM/RANDN

     Reference page in Doc Center
        doc randn

     Other functions named randn

        RandStream/randn


x =

  Columns 1 through 7

    5.5933    0.2220    0.5906    4.6591   -2.1557   -3.8703    3.2621

  Columns 8 through 14

    3.2022    2.2854    3.4901    5.2467    4.9113    0.2943    4.4299

  Columns 15 through 21

    2.5197    0.4834   -1.5799    3.9409    0.9391    2.1393   -0.3788

  Columns 22 through 28

   -2.6515    2.5148    1.8136    5.5971    4.4051    5.1599   -0.2466

  Columns 29 through 35

   -0.8090   -1.8073    3.4939   10.3672    4.1327   -0.3192    4.5099

  Columns 36 through 40

   -1.3850   -2.2734    4.1523   -0.3337    2.9480
```

first scale the standard normal by the desired standard deviation then shift by the desired mean

d)

```
x = 2*randn(1,40)+3
```

```
x =

  Columns 1 through 7

    5.8131    3.8022    4.8593   -0.2116    4.3231    7.2770    4.0823

  Columns 8 through 14

   -0.0818    2.5937    2.0001    3.7660    3.8241    3.8110    2.2724

  Columns 15 through 21

    1.8015    1.8208    4.7071   -0.7060    2.5854    3.5408    1.6945

  Columns 22 through 28

    3.9545    2.8574    1.1234    3.3227    2.4636    2.1803    1.5774

  Columns 29 through 35
```

```
     3.1229   -0.6923    2.2033    1.9129    1.1762    4.3054    1.5315

  Columns 36 through 40

     4.0813    4.9517    2.6863    3.5556    4.2790
```

first scale the standard normal by the desired standard deviation then shift by the desired mean

e) This script has 2 loops containing a `pause` statement. With the command window of figure window active, press any key to exit the pause and see the next plot (in the same figure window because of `hold on`)

You should see a sequence of 20 windows with increasing numbers of random (x,y) coordinates plotted in the unit square, then 20 windows with increasing numbers of (x,y) coordinates drawn from a standard normal distribution, then 4 histograms - 2 showing uniform random numbers, 2 showing normal random numbers.

f) I intended you to modify just the first loop

```
type demorandi
```

```
% Adapted from demorand.m by D. O'Leary

nn = 20;
mm = 30;

close all
hold on
title('Plot of random integers')
Xu = [];
for i=1:nn
x = randi(10,mm,2);
Xu = [Xu;x];
plot(x(:,1),x(:,2),'*');
pause
end

% Display histograms of the numbers

figure(2)
Xu = reshape(Xu,nn*mm*2,1);
histogram(Xu)
title(sprintf('Histogram of %d  random integers',nn*mm*2))

figure(3)
histogram(randi(10,100000,1))
title('Histogram of 100000  random integers')
```

Exercise set 3a

a) The best answer is to regard throwing 2 dice as a random experiment with 36 equally likely outcomes i.e. like a 36-sided die. This is what I intended.

```
type deMere2
```

```
function probDoubleSix = deMere2(numReps)
    numRolls= 24;
    numDoubleSixes = 0;
    for run = 1: numReps
        roll = randi(36,numRolls,1); % the random expt
        if any(roll==36)
            numDoubleSixes = numDoubleSixes + 1; % the quantity of interest
        end
    end
    probDoubleSix = numDoubleSixes/numReps; % the frequency of a 6
    % fprintf('Prob of a double 6 is %6.4f\n',probSix)
end
```

Most students did something more literal by throwing each die separately. Here is an example using this approach:

```
type deMere2a
```

```
function probDoubleSix = deMere2a(numReps)
    numRolls= 24;
    numDoubleSixes = 0;
    for run = 1: numReps
        roll1 = randi(6,numRolls,1); % the random expt
        roll2 = randi(6,numRolls,1);
        if any(roll1==6 & roll2==6) % note the use of element-wise logical AND &
            numDoubleSixes = numDoubleSixes + 1; % the quantity of interest
        end
    end
    probDoubleSix = numDoubleSixes/numReps; % the frequency of a 6
    % fprintf('Prob of a double 6 is %6.4f\n',probSix)
end
```

the other kind of slution

```
type deMere2b
```

```
function probDoubleSix = deMere2b(numReps)
    numRolls= 24;
    numDoubleSixes = 0;
    for run = 1: numReps
        roll1 = randi(6,numRolls,1); % the random expt
        roll2 = randi(6,numRolls,1);
        if any((roll1+roll2)==12) % add together and check with 12
            numDoubleSixes = numDoubleSixes + 1; % the quantity of interest
        end
    end
    probDoubleSix = numDoubleSixes/numReps; % the frequency of a 6
    % fprintf('Prob of a double 6 is %6.4f\n',probSix)
end
```

```
%b) Here is a typical driver

type deMereDriver
```

```
% demereDriver

for k=1:10
    prob2(k) =deMere1(100);
    prob4(k) =deMere1(10000);
%     prob2(k) =deMere2a(100);
%     prob4(k) =deMere2a(10000);
end
prob2 =reshape(prob2,2,5);
prob4=reshape(prob4,2,5);
disp(prob2);disp(prob4);
```

```
%c) Here is solution
type ruin
```

```
function [probRuin, meanTime ] = ruin( k,T,Ntrials )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

numRuin = 0; sumTime = 0;
p = 18/37; q=1-p;
for count = 1:Ntrials
    kevin = k; rolls = 0;
    while (kevin>0 && kevin < T)
        % alertative  way to do it is to using random integer between [1, 37]
        % then we can use the following two line code to replace if(rand(1) < p)
        % roll = randi(37,1);
        % if (roll < 19)
        if (rand(1)<  p)
            kevin = kevin+1;
        else
            kevin = kevin -1;
        end
        rolls = rolls + 1;
    end
    sumTime = sumTime + rolls;
    if kevin == 0
        numRuin = numRuin + 1;
    end
```

```
end
probRuin = numRuin/Ntrials;
meanTime = sumTime/Ntrials;

%
exactRuin = ((q/p)^T-(q/p)^k)/((q/p)^T-1);
approxMeanTime = k/(q-p);
fprintf('%10.5f %10.5f \n',exactRuin,approxMeanTime);
end
```

Published with MATLAB® R2019a