All LOCATION data refer to the following website where programs are stored.
www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html
An index of the Mathematica programs below is shown at the end of this document.

PROGRAM: CoinTosses
CALLING SEQUENCE: CoinTosses[n, print]
PARAMETERS: n - an integer, print - a Boolean variable (True or False)
SYNOPSIS:
- This program simulates n tosses of a fair coin, and prints the proportion of tosses that come up heads. If print = True, then the outcomes of the tosses (H/T) are also printed.

RETURNED VALUES:
- none

LOCATION:
Folder: Chapter 1
File: "CoinTosses.Chpt1.mat"

```
Clear[CoinTosses];
CoinTosses[n_, print_] :=
Block[
          {headcounter = 0,outputstring = ""},
    For[i = 1, i <= n, i++,
       If[(Random[] < .5),
         headcounter++;
         If[print,
           outputstring =
                StringInsert[outputstring, "H", i]
           ],
         If[print,
           outputstring =
                StringInsert[outputstring, "T", i]
           ]
         ]
       ];
    Print[outputstring];
    Print[" "];
    Print["The proportion of heads in ", n,
        " tosses is ", N[headcounter/n, 5]
        ]
   ]
```

Example for CoinTosses
CoinTosses[100, True]
HTHTHHHHTHTHHTTTHHTTTTHTTHTHTTTTTTHTTTHTHHHTHHHTTHHTHTTHHT
TTTHTTHTHTHTHTHHTHHTHTTTHHHHTTTHHTHTHTHTTTHT

The proportion of heads in 100 tosses is 0.46

PROGRAM: DeMere1
CALLING SEQUENCE: DeMere1[n, print]
PARAMETERS: n - an integer, print - a Boolean variable (True or False)
SYNOPSIS:
- This program simulates 4 rolls of a die, and determines whether a six has appeared (a "success"). It repeats this experiment n times, and prints the number of trials that resulted in a success. It also prints the proportion of trials that resulted in a success. Finally, if print = True, then the rolls are printed out.

RETURNED VALUES:
- none

LOCATION:
Folder: Chapter 1
File: "DeMere1&2.Chpt1.mat"

```
Clear[DeMere1];
DeMere1[n_, print_] :=
Block[
    {successcounter = 0,
     currentrollset = {}
    },
    For[i = 1, i <= n, i++,
       currentrollset = {};
       For[j = 1, j <= 4, j++,
          currentrollset =
            Append[currentrollset,
                Ceiling[6*Random[]]
               ];
         ];
       If[print, Print[currentrollset]
         ];
       If[Count[currentrollset, 6] > 0,
         successcounter++
         ];
      ];
    Print["number of successes = ",
        successcounter
       ];
    Print["proportion of successes = ",
        N[successcounter/n, 5]
       ]
   ]
```

Example for DeMere1

```
DeMere1[1000, False]
number of successes = 522
proportion of successes = 0.522
```

PROGRAM: DeMere2
CALLING SEQUENCE: DeMere2[n, m, print]
PARAMETERS: n, m - integers
print - a Boolean variable (True or False)
SYNOPSIS:
- This program simulates m rolls of two dice, and determines whether a double 6 has appeared (a "success"). It repeats this experiment n times, and prints the number of trials that resulted in a success. It also prints the proportion of trials that resulted in a success. Finally, if print = True, then the rolls are printed out.
RETURNED VALUES:
- none
LOCAT ION:
Folder: Chapter 1
File: "DeMere1&2.Chpt1.mat"

```
Clear[DeMere2];
DeMere2[n_, m_, print_] :=
Block[{successcounter = 0,
      currentrollset = {}
     },
     For[i = 1, i <= n, i++,
        currentrollset = {};
        For[j = 1, j <= m, j++,
           currentrollset =
             Append[currentrollset,
                   {Ceiling[6*Random[]],
                    Ceiling[6*Random[]]
                   }
                  ];
          ];
        If[print, Print[currentrollset]
          ];
        If[Count[currentrollset, {6, 6}] > 0,
          successcounter++
          ];
       ];
     Print["number of successes = ",
         successcounter
        ];
     Print["proportion of successes = ",
         N[successcounter/n, 5]
        ]
    ]
```

Example for DeMere2

```
DeMere2[1000, 24, False]
number of successes = 475
proportion of successes = 0.475
```

PROGRAM: RandomNumbers
CALLING SEQUENCE: RandomNumbers[n]
PARAMETERS: n - an integer
SYNOPSIS:
- This program generates and displays n random real numbers between 0 and 1.

RETURNED VALUES:
- none

LOCATION:
Folder: Chapter 1
File: "RandomNumbers.Chpt1.mat"

```
Clear[RandomNumbers];
RandomNumbers[n_] := Do[Print[Random[]],
                        {n}
                       ]
```

Example for RandomNumbers

```
RandomNumbers[3]
0.132088
0.459366
0.178864
```

PROGRAM: SpikegraphWithDots
CALLING SEQUENCE: SpikegraphWithDots[distributionlist, xmin, xmax, color, print]
PARAMETERS:
distributionlist - a distribution list
xmin, xmax - real numbers
color - a list of 3 color-specification real numbers
print - a Boolean variable (True or False)
SYNOPSIS:
- This program displays a graph of the distribution of x (where x has the distribution given in distributionlist) by drawing a spike of height p(x) at each x, and topping that spike with a dot of color color. If print = True, this graph is displayed. Otherwise, the display is (for the time being) suppressed. (If the graph has been suppressed, to see it at a later time type "Show[%#, DisplayFunction -> $DisplayFunction]", where # is the input number of the original call to SpikegraphWithDots.) The input distribution list is assumed to be in increasing order of the x-values. Important note: only values of x which fall in the user-defined interval [xmin, xmax] will be included in the graph. If not all values of x are included, and print = True, a warning is displayed. If print = False, no such warning will be given, even if the graph is later displayed.

RETURNED VALUES:
- none

LOCATION:

File: "Important Programs"

```
Clear[SpikegraphWithDots];
SpikegraphWithDots[distributionlist_, xmin_, xmax_,
                   color_, print_] :=
Block[{num = Length[distributionlist],
       j, k,
       linelist
      },
      linelist = Table[Line[{distributionlist[[i]],
                            {distributionlist[[i]][[1]], 0}}
                           ],
                       {i, 1, num}
                      ];
      pointlist = Table[Point[distributionlist[[i]]],
                        {i, 1, num}
                       ];
      j = 1;
      While[distributionlist[[j]][[1]] < xmin,
            linelist = Drop[linelist, 1];
            pointlist = Drop[pointlist, 1];
            j++
           ];
      k = num;
      While[distributionlist[[k]][[1]] > xmax,
            linelist = Drop[linelist, -1];
            pointlist = Drop[pointlist, -1];
            k--
           ];
      finallist = Join[linelist, pointlist];
      If[print,
         If[((distributionlist[[1]][[1]] < xmin)||
             (distributionlist[[num]][[1]] > xmax)),
            Print["Note:  some outcome values lie outside the
user-defined interval."]
           ];
         Show[Graphics[linelist],
              Graphics[{PointSize[.02],
                        RGBColor[color[[1]], color[[2]],
color[[3]]],
                        pointlist}
                       ],
              PlotRange -> All,
              Frame -> True
             ],
         Show[Graphics[linelist],
              Graphics[{PointSize[.02],
```

```
                              RGBColor[color[[1]], color[[2]],
color[[3]]],
                              pointlist}
                             ],
                  DisplayFunction->Identity,
                  PlotRange -> All,
                  Frame -> True
                 ]
          ]
      ]
```

PROGRAM: SimulateDiscreteVariable
CALLING SEQUENCE: SimulateDiscreteVariable[plist]
PARAMETERS:
plist - a probability list
SYNOPSIS:
- This program simulates an experiment which has outcomes x_1, x_2, ..., x_(Length[plist]) with probabilities plist[[1]], plist[[2]], ..., plist[[Length[plist]]], respectively. The program returns i, where x_i is the outcome of the experiment.

RETURNED VALUES:
- i , where x_i is the outcome of the experiment

LOCATION:
File: "Important Programs"

```
Clear[SimulateDiscreteVariable];
SimulateDiscreteVariable[plist_] :=
Block[{r, j = 1, subtotal = 0},
      r = Random[];
      While[subtotal <= r,
            subtotal = subtotal + plist[[j]];
            j++
           ];
      Return[j - 1]
      ]
```

PROGRAM: GeneralSimulation
CALLING SEQUENCE: GeneralSimulation[n, plist, m, print]
PARAMETERS: n, m – integers, plist - a probability list
print - a Boolean variable (True or False)
SYNOPSIS:
- This program simulates a general experiment in which the outcomes 1, 2, ..., n occur with probabilities p(1), p(2), ..., p(n). These probabilities are entered as a list in plist. The experiment is repeated m times, and the observed frequencies of the outcomes are printed. In addition, spike graphs of the observed data frequencies and plist data frequencies are displayed on the same set of axes;

the observed data spikes are topped with blue dots, and the plist data spikes are topped with red. If print = True, the outcomes are printed. Finally, the program returns a list of the n observed frequencies.

- Note: this program requires the programs "SimulateDiscreteVariable[plist]" and "SpikegraphWithDots[distributionlist, xmin, xmax, color, print]" be initialized.

RETURNED VALUES:

- a list of the n observed frequencies

LOCATION:

Folder: Chapter 1

File: "GeneralSimulation.Chpt1.mat"

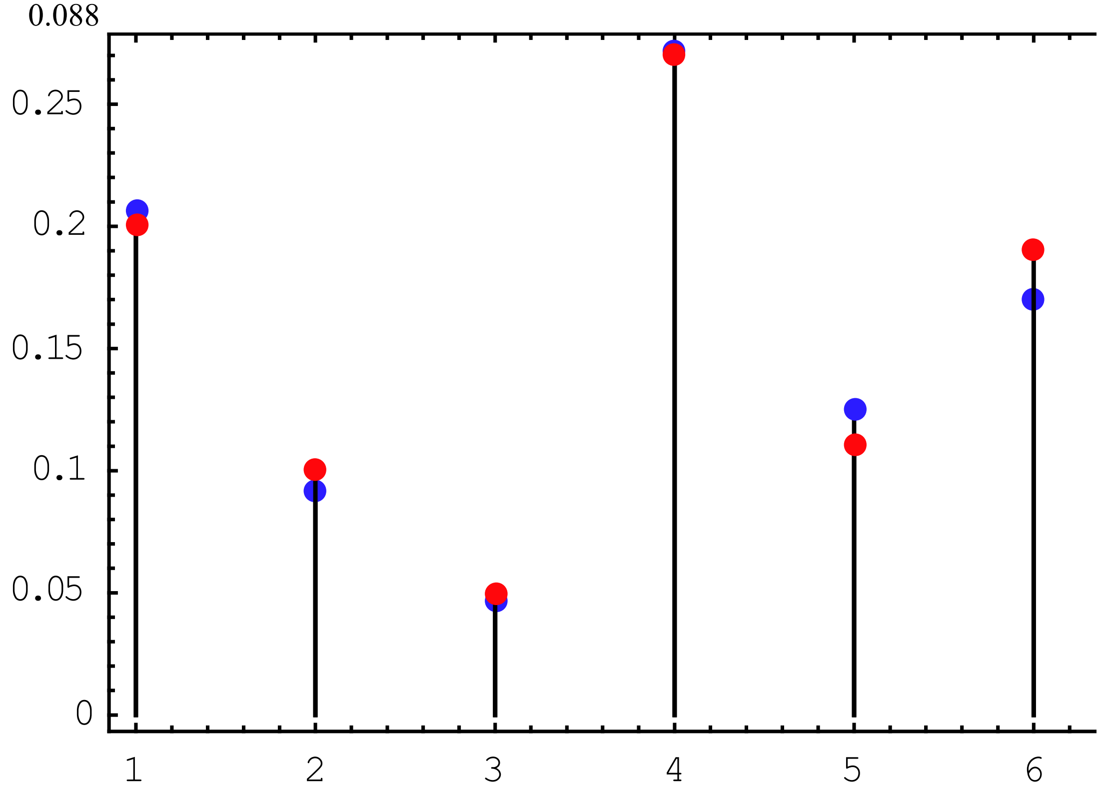
```
Clear[GeneralSimulation];
GeneralSimulation[n_, plist_, m_, print_] :=
Block[{x,
      observedfreqlist = Table[0, {j, 1, n}],
      j,
      subtotal,
      alist, blist
     },
     For[i = 1, i <= m, i++,
        w = SimulateDiscreteVariable[plist];
        observedfreqlist[[w]]++;
        If[print, Print[w]];
       ];
     Print[" "];
     Print["Outcome  p(k)    p*(k)"];
     Print[" "];
     For[k = 1, k <= n, k++,
        Print[k,  "        ",
             plist[[k]], "     ",
             N[observedfreqlist[[k]]/m, 5]
            ];
       ];
     alist = Table[{i, N[observedfreqlist[[i]]/m, 5]},
                 {i, 1, Length[observedfreqlist]}
                ];
     blist = Table[{i, plist[[i]]},
                 {i, 1, Length[plist]}
                ];
     g1 = SpikegraphWithDots[alist, 1, Length[alist],
                      {0.170, 0.110, 1.000}, False
                     ];
     g2 = SpikegraphWithDots[blist, 1, Length[blist],
                      {1.000, 0.030, 0.049}, False
                     ];
     Show[g1, g2, DisplayFunction -> $DisplayFunction];
     Return[N[observedfreqlist/m, 5]]
```

]

Example for GeneralSimulation

GeneralSimulation[7, {.2, .1, .05, .27, .11, .19, .08}, 1000,
False]

Outcome	p(k)	p*(k)
1	0.2	0.207
2	0.1	0.091
3	0.05	0.047
4	0.27	0.272
5	0.11	0.125
6	0.19	0.17
7	0.08	0.088



PROGRAM: MonteCarlo

CALLING SEQUENCE: contecarlo[n, f, xmin, xmax, ymax]

PARAMETERS:

n - an integer

f - the name of a pre-defined function of one variable

xmin, xmax, ymax - real numbers

SYNOPSIS:

- This program estimates the area under the input function f[x] and above the interval [xmin, xmax] by choosing n random points in the rectangle above the interval [xmin, xmax] and between the y-values 0 and ymax. The function f[x] is assumed to be non-negative on the interval [xmin, xmax], and is assumed to have a maximum value which does not exceed ymax. (Note: it is not necessary that the maximum value of f[x] = ymax.) The program returns its area estimate, and also plots the random points and the function f[x] on the interval [xmin, xmax].

- Keep in mind that the function f[x] should be defined before this program is called, and then the name of the function, namely f (or some other name, like Cos) should be given to this program. The expression for the function (such as x^2, for example) should not be given as a parameter.

RETURNED VALUES:

- none

LOCATION:

Folder: Chapter 2
File: "MonteCarlo.Chpt2.mat"

```
Clear[montecarlo];
montecarlo[n_, f_, xmin_, xmax_, ymax_] :=
Block[{boxarea = ymax*(xmax - xmin),

(*  Because of a bug in the Mathematica function Random, we
    must first take numerical approximations to the input values
    xmin, xmax, and ymax.
*)

       nymax = N[ymax],
       nxmax = N[xmax],
       nxmin = N[xmin],
       count = 0,
       randpoint,
       pointlist = {}
       },
      For[i = 1, i <= n, i++,
         randpoint = {Random[Real, {nxmin, nxmax}],
                  Random[Real, {0, nymax}]
                  };
         pointlist = Append[pointlist, randpoint];
         If[(randpoint[[2]] <= f[randpoint[[1]]]),
           count++
          ];
        ];
      Print[(count/n)*ymax*(xmax - xmin)//N];

(* In the following command, the first two graphics calls
are given the option DisplayFunction -> Identity so
that they are not plotted on the screen.  Then the Show
command is given the two plots with the DisplayFunction
option set back to the default value.  This results in
```

only one graph being shown on the screen, rather than
three.*)

```
      Show[{Plot[f[x], {x, xmin, xmax},
            DisplayFunction -> Identity],
            ListPlot[pointlist, DisplayFunction -> Identity]
           }, DisplayFunction -> $DisplayFunction,
           AspectRatio -> 1
          ];
     ]
```

Example for Montecarlo.

```
Clear[f];
f[x_] := x^2
montecarlo[100, f, 0, 1, 1]
0.35
```

Index of above programs:

CoinTosses --Generates a sequence of H's and T's and tells the percentage of H's
DeMere1 – generates sample fraction of wins for de Mere's first game
DeMere2 --generates sample fraction of wins for de Mere's second game
RandomNumbers –generates a sample of random numbers from (0,1)
SpikegraphWithDots --subroutine
SimulateDiscreteVariable --subroutine
GeneralSimulation --generates a sample from a specified discrete distribution—requires initialization of subroutines SimulateDiscreteVariable and SpikegraphWithDots.
Montecarlo –estimates area under a curve inside a rectangle