Illustrating the Use of Discrete Distributions

Initialization Code (optional)

Manipulate

```
Manipulate[
 Which[
  doPoisson == True, poisson[λ, connect, grid, verticalLines,
   doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile],

  doLogSeries == True, logSeries[θ, connect, grid, verticalLines,
   doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile],

  doBinomial == True, enableConnect = True; nBinomial = Round[nBinomial];
  binomial[nBinomial, pBinomial, connect, grid, verticalLines, doListPlot,
   dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile],

  doZipf == True, zipf[pZipf, connect, grid, verticalLines,
   doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile],

  doSkellam == True,
  First@{If[mu1Skellam == 0, mu1Skellam = 0.001];
    If[mu2Skellam == 0, mu2Skellam = 0.001]; skellam[mu1Skellam, mu2Skellam, connect, grid,
     verticalLines, doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile]
   },

  doBetaBinomial == True,
  First@
   {nBetaBinomial = Round[nBetaBinomial]; betaBinomial[αBetaBinomial, βBetaBinomial, nBetaBinomial, connect,
     grid, verticalLines, doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile]
   },
  doBetaNegativeBinomial == True,
  First@{nBetaNegativeBinomial = Round[nBetaNegativeBinomial];
    betaNegativeBinomial[αBetaNegativeBinomial, βBetaNegativeBinomial, nBetaNegativeBinomial, connect,
     grid, verticalLines, doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile]
   },
  doGeometric == True, geometric[pGeometric, connect, grid, verticalLines,
   doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile],
  doBernoulli == True, bernoulli[pBernoulli, connect, grid , verticalLines, doListPlot,
   dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile],
  doNegativeBinomial == True,
  First@{
    nNegativeBinomial = Round[nNegativeBinomial];
    If[pNegativeBinomial == 0, pNegativeBinomial = 0.01];
    negativeBinomial[nNegativeBinomial, pNegativeBinomial, connect, grid,
     verticalLines, doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile]
   },
  doDiscreteUniform == True,
  First@{If [max == min,  max = min + 1, If[max < min, max = min + 1]] ;
     discreteUniform[min, max, connect, grid, verticalLines,
      doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile]
   }
```

```
 ,
 doHyperGeometric == True, First@{
   If[nHyperGeometric > nTotalHyperGeometric, nHyperGeometric = nTotalHyperGeometric];
   If[nBlackHyperGeometric > nTotalHyperGeometric, nBlackHyperGeometric = nTotalHyperGeometric];
   nHyperGeometric = Round[nHyperGeometric];
   nTotalHyperGeometric = Round[nTotalHyperGeometric];
   nBlackHyperGeometric = Round[nBlackHyperGeometric];

   hyperGeometric[nHyperGeometric, nTotalHyperGeometric, nBlackHyperGeometric, connect, grid,
    verticalLines, doListPlot, dolabelValues, doSpaceBetweenBarChart, barChartColor, quantile]
  }
],

(*virtual controls to localized dynamics symbols*)
{{doPoisson, True}, None},
{{doBinomial, False}, None},
{{ doBernoulli, False}, None},
{{ doGeometric, False}, None},
{{doBetaBinomial, False}, None},
{{doBetaNegativeBinomial, False}, None},
{{doNegativeBinomial, False}, None},
{{doDiscreteUniform, False}, None},
{{doHyperGeometric, False}, None},
{{doZipf, False}, None},
{{doLogSeries, False}, None},
{{doSkellam, False}, None},

{{λ, 1.0}, .1, 30, .1, None},
{{pBinomial, .1}, None},
{{nBinomial, 40}, None},
{{pBernoulli, .4}, None},
{{pGeometric, .5}, None},
{{pNegativeBinomial, .26}, None},
{{nNegativeBinomial, 2}, None},
{{nHyperGeometric, 6}, None},
{{nTotalHyperGeometric, 30}, None},
{{nBlackHyperGeometric, 6}, None},
{{pZipf, 1.5}, None},
{{θ, .5}, None},
{{αBetaBinomial, .41}, None},
{{βBetaBinomial, .28}, None},
{{nBetaBinomial, 12}, None},
{{αBetaNegativeBinomial, 3}, None},
{{βBetaNegativeBinomial, .71}, None},
{{nBetaNegativeBinomial, 12}, None},
{{min, 2}, None},
{{max, 8}, None},

{{mu1Skellam, 2}, None},
{{mu2Skellam, 3}, None},

{{plotMaxX, 10}, None},
{{plotMaxY, 1}, None},
{{connect, True}, None},

{{grid, False}, None},
{{verticalLines, True}, None},
{{dolabelValues, True}, None},

{{doListPlot, False}, None},
{{doBarChart, True}, None},
{{doSpaceBetweenBarChart, True}, None},
{{barChartColor, Yellow}, None},
{{quantile, .95}, None},
```

```
Grid[{
  {
   (*FIRST ROW *)
   Grid[{{
       (* POISSON *)
       Grid[{
         {
          Grid[{{
              RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
                 doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
                 doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
               {True, False, False, False, False, False, False, False, False, False, False}],
              Style["Poisson", sz, Bold]
             }}]
         },
         {
          Grid[{{
              Style["λ", sz],
              Manipulator[Dynamic[λ], {0.1, 10, .1},
               Sequence[AppearanceElements → {"StepLeftButton",
                   "StepRightButton", "PlayPauseButton"}, ImageMargins → -2, ImageSize → Tiny],
               Enabled → Dynamic[(doPoisson == True) ], ContinuousAction → True
              ],
              Style[Dynamic[padIt2[λ, {2, 1}]], 12]
             }}]
         }
        }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}],

       (* Bernoulli *)
       Grid[{
         {
          Grid[{{
              RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
                 doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
                 doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
               {False, False, True, False, False, False, False, False, False, False, False, False}],
              Style["Bernoulli", sz, Bold]
             }}]
         },
         {
          Grid[{{
              Style["p", sz],
              Manipulator[Dynamic[pBernoulli], {0, 1, .025},
               Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
                 ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doBernoulli == True)]
              ],
              Style[Dynamic[padIt2[pBernoulli, {4, 3}]], 12]
             }}]
         }
        }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}],

       (* Geometric *)
       Grid[{
         {
          Grid[{{RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
                 doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
                 doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
               {False, False, False, True, False, False, False, False, False, False, False, False}],
              Style["Geometric", sz, Bold]
             }}]
         },
         {
          Grid[{{
```

```
            Style["p", sz],
            Manipulator[Dynamic[pGeometric], {0.1, 1, .01},
             Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
              ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doGeometric == True)]
            ],
            Style[Dynamic[padIt2[pGeometric, {3, 2}]], 12]
           }}]
       }
      }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}]
   }}, Alignment → Left]
},
{
 (* 2nd ROW *)
 Grid[{{

     (* Binomial *)
     Grid[{{
        Grid[{{
            RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
               doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
               doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
             {False, True, False, False, False, False, False, False, False, False, False, False}],
            Style["Binomial", sz, Bold]
           }}]
       },
       {Grid[{{
            Grid[{{
               Style["n", sz],
               Manipulator[Dynamic[nBinomial], {1, 50, 1},
                Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
                 ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doBinomial == True)]
               ],
               Style[Dynamic[padIt2[nBinomial, 2]], 12]
              }}]
           },
           {
            Grid[{{
               Style["p", sz],
               Manipulator[Dynamic[pBinomial], {0, 1, .05},
                Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton",
                    "PlayPauseButton"}, ImageMargins → -2, ImageSize → Tiny]],
               Style[Dynamic[padIt2[pBinomial, {3, 2}]], 12]
              }}],
           }
          }]
       }
      }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}
     ],

     (* Negative binomial *)
     Grid[{
       {
        Grid[{{
            RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
               doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
               doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
             {False, False, False, False, False, True, False, False, False, False, False, False}],
            Style["Negative binomial", sz, Bold]
           }}]
       },
       {
        Grid[{{
            Grid[{{
               Style["n", sz],
```

```
            Manipulator[Dynamic[nNegativeBinomial], {1, 10, 1}, Sequence[
              AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
              ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doNegativeBinomial == True)]],
            Style[Dynamic[padIt2[nNegativeBinomial, 2]], 12]
           }}]
       },
       {
        Grid[{{
            Style["p", sz],
            Manipulator[Dynamic[pNegativeBinomial], {0.1, 1, .1}, Sequence[
              AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
              ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doNegativeBinomial == True)]],
            Style[Dynamic[padIt2[pNegativeBinomial, {3, 2}]], 12]
           }}]
       }
      }]
    }
   }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}
  ],

  (*uniform   *)
  Grid[{
    {
     Grid[{{
         RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
            doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
            doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
          {False, False, False, False, False, False, True, False, False, False, False, False}],
         Style["Discrete Uniform", sz, Bold]
        }}]
    },
    {
     Grid[{{
         Style["min", sz],
         Manipulator[Dynamic[min], {-10, 10 - 1, 1},
          Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
           ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doDiscreteUniform == True)]],
         Style[Dynamic[padIt1[min, 2]], 12]
        }}
     ]
    },
    {
     Grid[{{
         Style["max", sz],
         Manipulator[Dynamic[max], {-10 + 1, 10, 1},
          Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
           ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doDiscreteUniform == True)]],
         Style[Dynamic[padIt1[max, 2]], 12]
        }}]
    }
   }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}
  ]
 }}, Alignment → Left
]
},
{
 (* 3rd ROW *)
 Grid[{{

   (* Beta binomial   *)
  Grid[{
    {
     Grid[{{
         RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
```

```
        doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
        doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
      {False, False, False, False, True, False, False, False, False, False, False, False}],
     Style["Beta Binomial", sz, Bold]
    }}]
  },
  {
   Grid[{{
     Grid[{{
       Style["α", sz],
       Manipulator[Dynamic[αBetaBinomial], {0.01, 2, 0.01},
        Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
         ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doBetaBinomial == True)]],
       Style[Dynamic[padIt2[αBetaBinomial, {3, 2}]], 12]
      }}]
    },
    {
     Grid[{{
       Style["β", sz],
       Manipulator[Dynamic[βBetaBinomial], {0.01, 1, 0.01},
        Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
         ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doBetaBinomial == True)]],
       Style[Dynamic[padIt2[βBetaBinomial, {3, 2}]], 12]
      }}]
    },
    {
     Grid[{{
       Style["n", sz],
       Manipulator[Dynamic[nBetaBinomial], {1, 20, 1},
        Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
         ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doBetaBinomial == True)]],
       Style[Dynamic[padIt2[nBetaBinomial, 2]], 12]
      }}]
    }
   }]
  }
 }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}
],

(* Beta negative binomial  *)
Grid[{
  {
   Grid[{{
     RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
       doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
       doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
      {False, False, False, False, False, False, False, False, False, False, True, False}],
     Style["Beta Negative Binomial", sz, Bold]
    }}]
  },
  {
   Grid[{
    {
     Grid[{{
       Style["α", sz],
       Manipulator[Dynamic[αBetaNegativeBinomial], {2.01, 6, 0.1}, Sequence[AppearanceElements →
         {"StepLeftButton", "StepRightButton", "PlayPauseButton"}, ImageMargins → -2, ImageSize →
         Tiny], ContinuousAction → False, Enabled → Dynamic[(doBetaNegativeBinomial == True)]],
       Style[Dynamic[padIt2[αBetaNegativeBinomial, {3, 2}]], 12]
      }}]
    },
    {
     Grid[{{
       Style["β", sz],
```

```
                Manipulator[Dynamic[βBetaNegativeBinomial], {0.01, 3.3, 0.1},
                 Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
                  ImageMargins → -2, ImageSize → Tiny], ContinuousAction → False,
                 Enabled → Dynamic[(doBetaNegativeBinomial == True)]],
                Style[Dynamic[padIt2[βBetaNegativeBinomial, {3, 2}]], 12]
               }}]
            },
            {
             Grid[{{
                Style["n", sz],
                Manipulator[Dynamic[nBetaNegativeBinomial], {1, 12, 1}, Sequence[AppearanceElements →
                   {"StepLeftButton", "StepRightButton", "PlayPauseButton"}, ImageMargins → -2, ImageSize →
                   Tiny], ContinuousAction → False, Enabled → Dynamic[(doBetaNegativeBinomial == True)]],
                Style[Dynamic[padIt2[nBetaNegativeBinomial, 2]], 12]
               }}]
            }
           }]
          }
         }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}],

         (* Hypergeometric   *)
        Grid[{
          {
           Grid[{{
              RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
                 doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
                 doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
               {False, False, False, False, False, False, False, True, False, False, False, False}
              ], Style["Hypergeometric", sz, Bold]
             }}]
          },
          {
           Grid[{
              {
               Style["n", sz],
               Manipulator[Dynamic[nHyperGeometric], {0, 40, 1}, Sequence[AppearanceElements →
                   {"StepLeftButton", "StepRightButton", "PlayPauseButton"}, ImageMargins → -2, ImageSize →
                   Tiny], ContinuousAction → False, Enabled → Dynamic[(doHyperGeometric == True)]],
               Style[Dynamic[padIt2[nHyperGeometric, 2]], 12]
              }}],
          },
          {
           Grid[{{Style["n_succ", 10],
              Manipulator[Dynamic[nBlackHyperGeometric], {0, 40, 1}, Sequence[AppearanceElements →
                   {"StepLeftButton", "StepRightButton", "PlayPauseButton"}, ImageMargins → -2, ImageSize →
                   Tiny], ContinuousAction → False, Enabled → Dynamic[(doHyperGeometric == True)]],
              Style[Dynamic[padIt2[nBlackHyperGeometric, 2]], 12]
             }}]
          },
          {
           Grid[{{
              Style["n_tot", 10],
              Manipulator[Dynamic[nTotalHyperGeometric], {1, 40, 1}, Sequence[AppearanceElements →
                   {"StepLeftButton", "StepRightButton", "PlayPauseButton"}, ImageMargins → -2, ImageSize →
                   Tiny], ContinuousAction → False, Enabled → Dynamic[(doHyperGeometric == True)]],
              Style[Dynamic[padIt2[nTotalHyperGeometric, 2]], 12]
             }}
           ]
          }
         }, Frame → True, FrameStyle → Gray, Alignment → Center
        ]
       }}, Alignment → Left
  ]
 },
```

```
{
 (* 4th ROW *)
 Grid[{{

     (* ZIPF *)
    Grid[{
      {
       Grid[{{RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
              doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
              doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
            {False, False, False, False, False, False, False, False, True, False, False, False}],
           Style["Zipf", sz, Bold]
          }}]
      },
      {
       Grid[{{
           Style["p", sz],
           Manipulator[Dynamic[pZipf], {0.01, 3, 0.01},
            Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
              ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doZipf == True)]],
           Style[Dynamic[padIt2[pZipf, {3, 2}]], 12]
          }}]
      }
     }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}],

    (* LOG *)
    Grid[{
      {
       Grid[{{
           RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
              doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
              doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
            {False, False, False, False, False, False, False, False, False, True, False, False}],
           Style["Logarithmic Series", sz, Bold]
          }}]
      },
      {
       Grid[{{
           Style["θ ", sz],
           Manipulator[Dynamic[θ], {0.01, 1 - 0.001, 0.01},
            Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
              ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doLogSeries == True)]],
           Style[Dynamic[padIt2[θ, {3, 2}]], 12]
          }}]
      }
     }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}],


    (* Skellam *)
    Grid[{
      {
       Grid[{{RadioButton[Dynamic[{doPoisson, doBinomial, doBernoulli,
              doGeometric, doBetaBinomial, doNegativeBinomial, doDiscreteUniform,
              doHyperGeometric, doZipf, doLogSeries, doBetaNegativeBinomial, doSkellam}],
            {False, False, False, False, False, False, False, False, False, False, False, True}],
           Style["Skellam", sz, Bold]
          }}]
      },
      {
       Grid[{
         {
          Grid[{{
              Style[\"μ1\", sz],
              Manipulator[Dynamic[mu1Skellam], {0, 5, .1},
```

```
                Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
                  ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doSkellam == True)]],
                Style[Dynamic[padIt2[mu1Skellam, {2, 1}]], 12]
               }}]
            },
            {
             Grid[{{
                 Style["μ2", sz],
                 Manipulator[Dynamic[mu2Skellam], {0, 5, .1},
                  Sequence[AppearanceElements → {"StepLeftButton", "StepRightButton", "PlayPauseButton"},
                    ImageMargins → -2, ImageSize → Tiny], Enabled → Dynamic[(doSkellam == True)]],
                 Style[Dynamic[padIt2[mu2Skellam, {2, 1}]], 12]
                }}]
            }
           }]
         }
        }, Frame → True, FrameStyle → Gray, Alignment → Center, ItemSize → {itemSize, 1}]
     }}, Alignment → Left]
},
{
 (* PLOT OPTIONS CONTROLS *)
 Grid[{{

    Grid[{
      {Style["annotate", sz2]},
      {Checkbox[Dynamic[dolabelValues]]}
     }, Frame → True, FrameStyle → Gray, Alignment → Center
    ],

    Grid[{
      {
       Grid[{
         {Style["ListPlot", sz2]},
         {
          RadioButton[Dynamic[{doListPlot, doBarChart}], {True, False}]
         }
        }],

       Grid[{
         {Checkbox[Dynamic[connect], Enabled → doListPlot], Style["joined", sz2]},
         {Checkbox[Dynamic[grid], Enabled → doListPlot], Style["grid", sz2]},
         {Checkbox[Dynamic[verticalLines], Enabled → doListPlot], Style["vertical lines", sz2]}
        }]
      }
     }, Frame → True, FrameStyle → Gray, Alignment → Center
    ],

    Grid[{
      {
       Grid[{
         {Style["BarChart", sz2]},
         {RadioButton[Dynamic[{doListPlot, doBarChart}], {False, True}]}
        }]
       ,
       Grid[{
         {Checkbox[Dynamic[doSpaceBetweenBarChart], Enabled → doBarChart]},
         {Style["bars space", sz2]}
        }]
       ,
       Grid[{
         {
          PopupMenu[Dynamic[barChartColor], {Red → Style["Red", sz], LightRed → Style["Light Red", sz],
            Green → Style["Green", sz], LightGreen → Style["Light Green", sz], Yellow →
             Style["Yellow", sz], Blue → Style["Blue", sz], LightBlue → Style["Light Blue", sz], Black →
```

```
                Style["Black", sz], Gray → Style["Gray", sz], LightGray → Style["Light Gray", sz], Cyan →
                 Style["Cyan", sz], LightCyan → Style["Light Cyan", sz], Magenta → Style["Magenta", sz],
                LightMagenta → Style["Light Magenta", sz], Brown → Style["Brown", sz], LightBrown →
                 Style["Light Brown", sz], Orange → Style["Orange", sz], LightOrange → Style["Light Orange",
                  sz], Pink → Style["Pink", sz], LightPink → Style["Light Pink", sz]}, Enabled → doBarChart]
            },
            {
             Style["bar color", sz2]
            }
           }]
        }}, Frame → True, FrameStyle → Gray, Alignment → Center
      ],

      Grid[{
        {
         PopupMenu[Dynamic[quantile], {.99 → Style["99%", sz], .98 → Style["98%", sz],
           .97 → Style["97%", sz], .96 → Style["96%", sz], .95 → Style["95%", sz], .90 → Style["90%", sz],
           .75 → Style["75%", sz], .50 → Style["50%", sz], .25 → Style["25%", sz], .15 → Style["15%", sz],
           .05 → Style["5%", sz], .025 → Style["2.5%", sz], .01 → Style["1%", sz]}]
        },
        {
         Style["Quantile", sz2]
        }
       }, Frame → True, FrameStyle → Gray, Alignment → Left
      ]
     }}, Spacings → {.1, .1}](*closes row for all plot options*)
  }}, Spacings → {0, .05}, Alignment → Center],

ControlPlacement → Top,
FrameMargins → 0,
ImageMargins → 0,
AutorunSequencing → {13},
Alignment → Center,

Initialization :→ (

  cpfm = 2; (*ontrolPanelFrameMargin*)
  sz = 11; (*size of labels for controls, see above *)
  sz2 = 11; (*size of labels for controls, see above *)
  itemSize = 13;

  integerStrictPositive = (IntegerQ[#] && # > 0 &);
  integerPositive = (IntegerQ[#] && # ≥ 0 &);
  numericStrictPositive = (Element[#, Reals] && # > 0 &);
  numericPositive = (Element[#, Reals] && # ≥ 0 &);
  numericStrictNegative = (Element[#, Reals] && # < 0 &);
  numericNegative = (Element[#, Reals] && # ≤ 0 &);
  bool = (Element[#, Booleans] &);
  numeric = (Element[#, Reals] &);
  integer = (Element[#, Integers] &);

  padIt1[v_?numeric, f_List] :=
   AccountingForm[v, f, NumberSigns → {"-", "+"}, NumberPadding → {"0", "0"}, SignPadding → True];
  padIt1[v_?numeric, f_Integer] := AccountingForm[Chop[v], f, NumberSigns → {"-", "+"},
    NumberPadding → {"0", "0"}, SignPadding → True];
  padIt2[v_?numeric, f_List] := AccountingForm[v, f, NumberSigns → {"", ""},
    NumberPadding → {"0", "0"}, SignPadding → True];
  padIt2[v_?numeric, f_Integer] := AccountingForm[Chop[v], f, NumberSigns → {"", ""},
    NumberPadding → {"0", "0"}, SignPadding → True];
```

```
(*-----------------------------------*)
(*   formats number for label values *)
(*-----------------------------------*)
formatNumberForLabel[num_] := Module[{n, m},
  n = RealDigits[num] [[2]];

  If[n ≤ 0,
   If[n == -5, m = 0, m = Round[num, 10^-5]],
   m = Round[num, 10^-1];
  ];

  If[ m == 0, "", StringJoin["  ", ToString[ If[m == 1, m = Round[m], N[m] ] ]]]
 ];

(*---------------------------------------*)
(*  called to make label values for PDF *)
(*---------------------------------------*)
buildLabelValuesForPDF[tbl_, leftSupport_, rightSupport_] :=
 Module[{maxNumberOfLabels = 18, labelValues, i, incr, center, dir, z},

  i = rightSupport - leftSupport + 1;
  incr = Floor[i / maxNumberOfLabels] + 1;

  labelValues = Table[
    {
     z = formatNumberForLabel[ tbl[[ i, 2]] ];

     If[z == "  1",
      {
       z = "1";
       dir = {1, 0};
       center = {0, -1}
      },
      {
       dir = {0, 1};
       center = {-1, 0}
      }
     ];

     Text[  z, {tbl[[ i, 1 ]], tbl[[ i, 2]] } , center, dir]
    },
    {i, 1, Length[tbl], incr }
   ]

 ];

(*---------------------------------------*)
(*  called to make label values for CDF *)
(*---------------------------------------*)
buildLabelValuesForCDF[tbl_, leftSupport_, rightSupport_] :=
 Module[{maxNumberOfLabels = 18, labelValues, i, incr, center, dir, z},

  i = rightSupport - leftSupport + 1;
  incr = Floor[i / maxNumberOfLabels] + 1;

  labelValues = Table[
    {
     z = formatNumberForLabel[ tbl[[ i, 2]] ];

     If[z == "  1",
      {
       z = "1";
       dir = {1, 0};
       center = {0, -1}
```

```
      },
      {
       dir = {0, 1};
       center = {-1, 0}
      }
     ];

      Text[  z, {tbl[[ i, 1 ]] + .5, tbl[[ i, 2]] } , center, dir]
    },
    {i, 1, Length[tbl], incr }
   ]

 ];
(*-------------------------------------------------------*)
(*  called to make plots from most of the distributions *)
(*-------------------------------------------------------*)
makePlots[distribution_, cdfLimits_, pdfAxesOrigion_, cdfAxesOrigin_, ticksForPdf_,
  ticksForCDF_, connect_, tbl_, grid_, label_, leftSupport_, rightSupport_, doListPlot_,
  verticalLines_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, cdflabel_] :=
 Module[{k, pdf, cdf, g, t, labelValues, max, cdftbl, i, lbls,
   cdft = Style["cumulative distribution function", 12], pmft = Style["probability mass function", 12],
   plotWidth = 260, plotHeight = 150, plotImagePadding = {{28, 12}, {20, 18}}},

  max = Max[ tbl[[ leftSupport ;; rightSupport , 2 ]] ];

  If[dolabelValues, labelValues = buildLabelValuesForPDF[tbl, leftSupport, rightSupport]];

  If[doListPlot,
   {
    g = Graphics[{Red, Line[  tbl[[ leftSupport ;; rightSupport]]  ] } ];

    pdf = ListPlot[tbl,
      If[verticalLines, {Filling → Axis, FillingStyle → Thin}, {Filling → None}],
      Frame → True,
      AxesOrigin → pdfAxesOrigion,
      FrameTicks → {{Automatic, None}, {ticksForPdf, None}},
      TicksStyle → Small,
      PlotLabel → Grid[{{pmft}, {label}}, Spacings → 0. Alignment → Center],
      PlotRange → {{-0.5, Automatic}, {0, 1.5 max}},
      ImagePadding → plotImagePadding,
      ImageSize → {plotWidth, plotHeight},
      AspectRatio → .3,
      If[grid, {GridLines → Automatic, GridLinesStyle → Directive[Red, Dashed]}, GridLines → None],
      If[dolabelValues,
       Sequence[{Axes → {True, False}, Epilog → {labelValues}}], ImagePadding → plotImagePadding]
     ];

    If[connect, pdf = Show[{pdf, g}] ];

   },
   {
    t = Table[{1, tbl[[i, 2]]  }, {i, 1, Length[tbl], 1}];
    lbls = Table[AccountingForm[N@t[[i, 2]], {3, 3}], {i, Length[t]}];
    placed = Table[{i - 1 + 0.5, 3 * tbl[[i, 2]]}, {i, 1, Length[tbl]}];

    pdf = RectangleChart[t,
      ChartStyle -> barChartColor,
      BarSpacing → If[doSpaceBetweenBarChart, Automatic, None],
      ImageSize → {plotWidth, plotHeight},
      Frame → None,
      BarSpacing → None,
      TicksStyle → Small,
      AspectRatio → .3,
      PlotLabel → Grid[{{pmft}, {label}}, Spacings → 0. Alignment → Center],
```

```
      ImagePadding → plotImagePadding
     ]
   }
  ];

  If[dolabelValues,
   {
    cdftbl = Table[{k, N@CDF[distribution, k]}, {k, cdfLimits[[1]], cdfLimits[[2]]  }];
    labelValues = buildLabelValuesForCDF[cdftbl, 1, Length[cdftbl]];
   }
  ];

  t = {k, cdfLimits[[1]], cdfLimits[[2]] + 1};

  cdf = Plot[CDF[distribution, k], Evaluate[t],
    Frame → True,
    ImagePadding → plotImagePadding,
    TicksStyle → Small,
    ImageSize → {plotWidth, plotHeight},
    PlotRange → {Automatic, If[Not[dolabelValues], {0, 1.1}, {0, 1.5}]},
    AxesOrigin → cdfAxesOrigin,
    FrameTicks → {{Automatic, None}, {ticksForCDF, None}},
    PlotLabel → Grid[{{cdft}, {cdflabel}}, Spacings → 0. Alignment → Center],
    PlotStyle → {Black, Thick},
    Axes → {True, False},
    AspectRatio → .3,
    Evaluate[
     If[grid, {GridLines → Automatic, GridLinesStyle → Directive[Red, Dashed]}, GridLines → None]],
    Evaluate[If[dolabelValues, Sequence[{Axes → {True, False}, Epilog → {labelValues}}],
      ImagePadding → plotImagePadding]]
   ];

  {pdf, cdf}
 ];

(*---------------------------------------*)
(*  find mean and variance               *)
(*---------------------------------------*)
getMeanVar[p_] := Module[{mean, var},

  mean = Mean[p];
  var = Variance[p];
  (*var=NumberForm[var,{9,7},NumberPadding→{" ","0"}];*)

  {mean, var}
 ];

(*-------------------------------------------------*)
(*  a way to guess a good x-axis limit for CDF    *)
(* inverse CDF does not work too well for all     *)
(*-------------------------------------------------*)
getLimit[y_, var_] := Module[{to = 5},
  While[(1 - (y /. var → to)) > .01, to = to + 10];
  to
 ];

(*-------------------------------------------------------*)
(*  another version of the above using specified delta  *)
(*-------------------------------------------------------*)
getLimit[y_, var_, delta_] := Module[{to = 5},
  While[(1 - (y /. var → to)) > delta, to = to + 10];
  to
 ];
```

```
(*-------------------------------------------------------------*)
(*  another version of the above using specified delta and jump *)
(*-------------------------------------------------------------*)
getLimit[y_, var_, delta_, inc_] := Module[{to = 5},
  While[(1 - (y /. var → to)) > delta, to = to + inc];
  to
 ];

(*-----------------------------------------------*)
(*  a way to guess good ticks                    *)
(*-----------------------------------------------*)
getTicks[from_, to_] := Module[{},
  If[to - from > 10, Range[from, to, Round[(to - from) / 10]], Range[from, to, 1]]
 ];

(*-------------------------------------------*)
(* helper function for formatting            *)
(*-------------------------------------------*)
padIt1[v_, f_List] :=
 AccountingForm[Chop[v] , f, NumberSigns → {"-", "+"}, NumberPadding → {"0", "0"}, SignPadding → True];

(*-------------------------------------------*)
(* helper function for formatting            *)
(*-------------------------------------------*)
padIt2[v_, f_List] :=
 AccountingForm[Chop[v] , f, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}, SignPadding → True];

(*-----------------------------------*)
(*  make labels                      *)
(*-----------------------------------*)
makeLabels[mean_, var_, skew_, kurtosis_, vquantile_] := Module[{label, cdfLabel},
  label = Style[Text@Grid[{
        {Style["mean =", 12], padIt1[N@mean, {4, 2}],
         Spacer[6], Style["variance =", 12], padIt1[N@var, {4, 2}]
        }}, Spacings → 0, Frame → None, Alignment → Center], 11];

  cdfLabel = Style[Text@Grid[
       {
        {Style["quantile = ", 11], padIt2[N@vquantile, {4, 2}]
        },
        {Style["skewness = ", 11],
         padIt1[N@skew, {4, 2}], Style["kurtosis ="], padIt1[N@kurtosis, {4, 2}]
        }

       }, Spacings → {.5, .1}, Frame → None, Alignment → Center
      ], 11];

  {label, cdfLabel}
 ];

(*-------------------------------------------------------------------*)
(*      S T A R T      O F     D I S T R I B I U T I O N S          *)
(*-------------------------------------------------------------------*)

(*----------------------*)
(*     BINOMIAL         *)
(*----------------------*)
binomial[n_, p_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, to, pdf, cdf, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@
   {
    dist = BinomialDistribution[n, p];
```

```
    {mean, var} = getMeanVar[dist];
    If[p == 0 || p == 1, {skew = Infinity; kurtosis = Infinity},
     {
      skew = N[Skewness[dist]];
      kurtosis = N[Kurtosis[dist]];
     }
    ];

    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    to = getLimit[CDF[dist, x], x];
    from = 0;

    tbl = Table[{k, PDF[dist, k]}, {k, from, to}];

    {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {-1, 0},
      Automatic, getTicks[-2, to], connect, tbl, grid, label, 1, Length[tbl], doListPlot,
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    (*label="X: Number of successes in n independent trials. Trial has p chance of success";
    data="mean: np="<>ToString[mean]<>"\tvariance: n(1-p)p="<>ToString[var];
    label=Style[data,Bold];
    *)

    Row[{pdf, cdf}]
   }
 ];

(*----------------------*)
(*    BERNOULLI         *)
(*----------------------*)
bernoulli[p_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, cdf, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k, label, pdf},

  First@{
    dist = BernoulliDistribution[p];
    {mean, var} = getMeanVar[dist];

    If[p == 0 || p == 1,
     {
      skew = Infinity;
      kurtosis = Infinity
     },
     {
      skew = N[Skewness[dist]];
      kurtosis = N[Kurtosis[dist]];
     }
    ];

    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];
    tbl = Table[{k, PDF[dist, k]}, {k, 0, 1}];

    {pdf, cdf} = makePlots[dist, {0, 2}, {0, 0}, {-1, 0}, {0, 1}, {-1, 0, 1, 2}, connect, tbl, grid, label, 1,
      2, doListPlot, verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    Row[{pdf, cdf}]
   }
 ];

(*----------------------------*)
```

```
(*    NEGATIVE BINOMIAL       *)
(*----------------------------*)
negativeBinomial[r_, p_, connect_, grid_, verticalLines_, doListPlot_,
  dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] := Module[
  {mean, var, label, from, to, pdf, cdf, x, dist, tbl, tblLines, cdfLabel, skew, kurtosis, vquantile, k},
  First@{
    dist = NegativeBinomialDistribution[r, p];
    {mean, var} = getMeanVar[dist];

    If[p == 1, {skew = Infinity; kurtosis = Infinity},
     {
      skew = N[Skewness[dist]];
      kurtosis = N[Kurtosis[dist]];
     }
    ];

    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    to = getLimit[CDF[dist, x], x, 0.1];
    from = 0;

    tbl = Table[PDF[dist, k], {k, from, to}];
    tblLines = Table[{k + 1, tbl[[k + 1]]}, {k, from, to}];

    {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {-1, 0}, Automatic,
      Automatic, connect, tblLines, grid, label, 1, Length[tblLines], doListPlot,
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    (*label="X: Total number of independent trials needed
       to obtain r successes\nwhen each trail has p chance of success";
    data="mean: r(1-p)/p="<>ToString[mean]<>"\tvariance: r(1-p)/p^2="<>ToString[var];
    label=Style[data,Bold];
    *)

    Row[{pdf, cdf}]
   }
 ];

(*--------------------*)
(*    GEOMETRIC       *)
(*--------------------*)
geometric[p_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, from, to, label, pdf, cdf, x, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@{
    dist = GeometricDistribution[p];
    {mean, var} = getMeanVar[dist];

    If[p == 1, {skew = Infinity; kurtosis = Infinity},
     {
      skew = N[Skewness[dist]];
      kurtosis = N[Kurtosis[dist]];
     }
    ];

    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];
    to = getLimit[CDF[dist, x], x];
    from = 0;
```

```
    tbl = Table[{k, PDF[dist, k]}, {k, from, to}];


    {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {-1, 0}, Automatic,
      If[p < .2, Automatic, getTicks[-2, to]], connect, tbl, grid, label, 1, Length[tbl],
      doListPlot, verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    (*label="X: The total number of trials up to and including first success";
    data="mean: 1/p-1="<>ToString[mean]<>"\tvariance:(1-p)/p^2="<>ToString[var];
   label=Style[data,Bold];
    *)
    Row[{pdf, cdf}]
   }
  ];

(*--------------------*)
(*   HYPER GEOMETRIC  *)
(*--------------------*)
hyperGeometric[n_, nTotal_, nBlack_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, to, pdf, cdf, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@{
    dist = HypergeometricDistribution[n, nBlack, nTotal];
    If[nTotal == 1, var = Indeterminate, var = N[Variance[dist] ]];
    mean = N[Mean[dist]];

    If[n == 1 && nBlack == 1 && nTotal ≤ 3,
     {
      kurtosis = Infinity;
      If[nTotal == 3, skew = N[Skewness[dist]], skew = Infinity]
     },
     If[ n == 0 || nBlack == 0  || n == nTotal || nBlack == nTotal,
       {
        skew = Infinity;
        kurtosis = Infinity;
       },
       {
        skew = N[Skewness[dist]];
        kurtosis = N[Kurtosis[dist]];
       }
      ];
    ];

    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    from = 0;
    to = Min[{n, nBlack}];
    tbl = Table[{k, PDF[dist, k]}, {k, from, to}];

    {pdf, cdf} = makePlots[dist, {from, to}, {0, 0}, {-1, 0},
      Automatic, Automatic, connect, tbl, grid, label, 1, Length[tbl], doListPlot,
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    (*label=
     "X: Number of black balls removed when m balls are removed\nfrom urn which contains n balls";
    data="mean ="<>ToString[mean]<>"\tvariance ="<>ToString[var];
   label=Style[data,Bold];
    *)
    Row[{pdf, cdf}]
   }
```

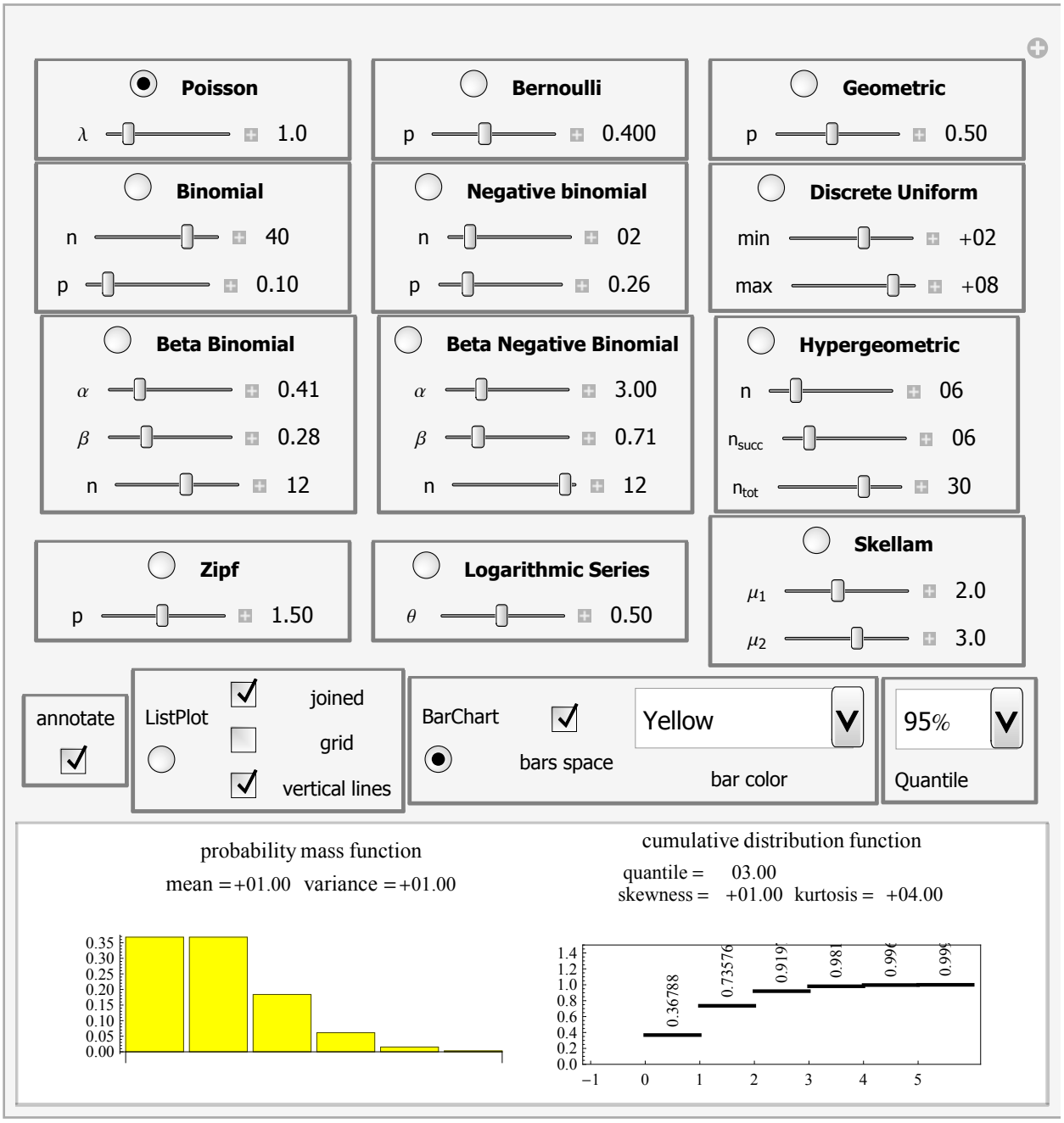
```
 ];

(*--------------------*)
(*   DISCRETE         *)
(*--------------------*)
discreteUniform[min_, max_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, to, pdf, cdf, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@{

    dist = DiscreteUniformDistribution[{min, max}];
    {mean, var} = getMeanVar[dist];
    skew = N[Skewness[dist]];

    If[min == max, kurtosis = Infinity, kurtosis = N[Kurtosis[dist]] ];

    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];
    from = min - 2;  to = max + 2;

    tbl = Table[{k, PDF[dist, k]}, {k, from, to}];

    {pdf, cdf} = makePlots[dist, {from, to}, {0, 0}, {0, 0},
      Automatic, Automatic, connect, tbl, grid, label, 3, Length[tbl] - 2, doListPlot,
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    (*
    data="mean: max+min/2 ="<>ToString[mean]<>"\tvariance: 1/12 (-1+(1+max-min)^2)="<>ToString[var];
    label="X: An integer between minumum and maximum";
    label=Style[data,Bold];
    *)

    Row[{pdf, cdf}]
   }
 ];

(*--------------------*)
(*     ZIPF           *)
(*--------------------*)
zipf[p_, connect_, grid_, verticalLines_, doListPlot_,
  dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, to, pdf, cdf, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@{
    dist = ZipfDistribution[p];
    {mean, var} = getMeanVar[dist];

    skew = N[Skewness[dist]];
    kurtosis = N[Kurtosis[dist]];

    If[p < .1, vquantile = Infinity, vquantile = Quantile[dist, quantile]];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    to = 15; (*getLimit[CDF[ZipfDistribution[p],x],x];*)
    from = 0;

    tbl = Table[{k, PDF[dist, k]}, {k, to}];


    {pdf, cdf} = makePlots[dist, {from, to}, {0, 0}, {0, 0},
      Automatic, Automatic, connect, tbl, grid, label, 1, Length[tbl], doListPlot,
```

```
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

    (*
    data="mean:Zeta[p]/Zeta[1+p]="<>ToString[mean]<>"\tvariance:Zeta[p-1]/Zeta[p+1]-Zeta[p]^2/Zeta[1+p]^2="<>ToString[var];
    label="X: Frequency of occurance of object with this rank";
   label=Style[data,Bold];
    *)
    Row[{pdf, cdf}]
   }
 ];

(*--------------------*)
(*     BETA BINOMIAL  *)
(*--------------------*)
betaBinomial[α_, β_, n_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, to, k, pdf, cdf, x, dist, tbl, cdfLabel, skew, kurtosis, vquantile},
  First@{
    dist = BetaBinomialDistribution[α, β, n];
    {mean, var} = getMeanVar[dist];
    skew = N[Skewness[dist]];
    kurtosis = N[Kurtosis[dist]];
    vquantile = Quantile[dist, quantile];

    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    to = getLimit[CDF[dist, x], x];
    from = 0;

    tbl = Table[{k, PDF[dist, k]}, {k, from, to}];

    {pdf, cdf} =
     makePlots[dist, {from, to}, {0, 0}, {-1, 0}, Automatic, getTicks[-2, to], connect, tbl, grid, label,
      1, to, doListPlot, verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];


    (*label="X: Number of successes in n trials where the probability of success\nin
       each trial follows a beta distribution with shape parameters α and β";
    data="mean:n α/(α+β) ="<>ToString[mean]<>"\tVar:n α β (n+α+β)/((α+β)^2 (1+α+β))="<>ToString[var];
   label=Style[data,Bold];
    *)

    Row[{pdf, cdf}]
   }
 ];

(*---------------------------------------*)
(*     BETA Negative BINOMIAL            *)
(*---------------------------------------*)
betaNegativeBinomial[α_, β_, n_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, pdflabel, cdfLabel, from, to, k, pdf, cdf, x, dist, tbl, kurtosis, skew, vquantile},

  dist = BetaNegativeBinomialDistribution[α, β, n];
  {mean, var} = getMeanVar[dist];
  vquantile = Quantile[dist, quantile];
  skew = N[Skewness[dist]];
  kurtosis = N[Kurtosis[dist]];
```

```
  {pdflabel, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

  to = getLimit[CDF[dist, x], x];

  k = 1;
  While[ CDF[dist, k] < .9 && k ≤ 10, k = k + 1] ;

  If[ CDF[dist, k] < .9,
   While[ CDF[dist, k] < .9 && k ≤ 100, k = k + 5]
  ];

  If[ CDF[dist, k] < .9,
   While[ CDF[dist, k] < .9 && k ≤ 10^3, k = k + 10]
  ];

  If[ CDF[dist, k] < .9,
   While[ CDF[dist, k] < .9 && k ≤ 10^4, k = k + 100]
  ];

  to = k;

  from = 0;

  tbl = N@Table[{k, PDF[dist, k]}, {k, from, to}];

  {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {-1, 0}, Automatic,
    getTicks[-2, to], connect, tbl, grid, pdflabel, 1, Length[tbl], doListPlot,
    verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

  (*label="X: Number of arrivals when average rate of arrival is λ";*)
  Row[{pdf, cdf}]

 ];

(*------------------*)
(*    LOG           *)
(*------------------*)
logSeries[θ_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, to, pdf, cdf, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@{dist = LogSeriesDistribution[θ];

    If[θ == 0, {skew = Infinity; kurtosis = Infinity},
     {
      skew = N[Skewness[dist]];
      kurtosis = N[Kurtosis[dist]];
     }
    ];

    {mean, var} = getMeanVar[dist];
    vquantile = Quantile[dist, quantile];
    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    (*to=getLimit[CDF[dist,x],x];*)
    to = 25;
    from = 1;

    tbl = Table[{k, PDF[dist, k]}, {k, to}];

    {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {0, 0},
      Automatic, getTicks[-1, to], connect, tbl, grid, label, 1, Length[tbl], doListPlot,
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];
```

```
    (*label="X: represents a logarithmic series distribution with parameter θ";*)
    (*data="mean: -θ/((1-θ) Log[1-θ])="<>ToString[mean]<>"\tvariance: -θ (1+θ/Log[1-θ])/((1-θ)^2 Log[1-θ])="<>ToString[var];
    label=Style[data,Bold];*)

    Row[{pdf, cdf}]
   }
  ];

(*--------------*)
(*    Skellam   *)
(*--------------*)
skellam[mu1_, mu2_, connect_, grid_, verticalLines_,
  doListPlot_, dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{cdfLabel, mean, var, pdflabel, from, to, pdf, cdf, dist, tbl, kurtosis, skew, vquantile
  },

  dist = SkellamDistribution[mu1, mu2];
  {mean, var} = getMeanVar[dist];
  vquantile = Quantile[dist, quantile];
  skew = N[Skewness[dist]];
  kurtosis = N[Kurtosis[dist]];

  {pdflabel, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

  from = 0;
  to = getLimit[CDF[dist, x], x];

  tbl = N@Table[{k, PDF[dist, k]}, {k, from, to}];

  {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {-1, 0}, Automatic,
    getTicks[-2, to], connect, tbl, grid, pdflabel, 1, Length[tbl], doListPlot,
    verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

  (*label="X: Number of arrivals when average rate of arrival is λ";*)
  Row[{pdf, cdf}]

 ];
(*--------------*)
(*    POISSON   *)
(*--------------*)
poisson[λ_, connect_, grid_, verticalLines_, doListPlot_,
  dolabelValues_, doSpaceBetweenBarChart_, barChartColor_, quantile_] :=
 Module[{mean, var, label, from, pdf, cdf, x, to, dist, tbl, cdfLabel, skew, kurtosis, vquantile, k},
  First@{

    dist = PoissonDistribution[λ];

    {mean, var} = getMeanVar[dist];
    vquantile = Quantile[dist, quantile];
    skew = N[Skewness[dist]];
    kurtosis = N[Kurtosis[dist]];

    {label, cdfLabel} = makeLabels[mean, var, skew, kurtosis, vquantile];

    from = 0;
    to = getLimit[CDF[dist, x], x];

    tbl = Table[{k, PDF[dist, k]}, {k, from, to}];

    {pdf, cdf} = makePlots[dist, {0, to}, {0, 0}, {-1, 0},
      Automatic, getTicks[-2, to], connect, tbl, grid, label, 1, Length[tbl], doListPlot,
```

```
      verticalLines, dolabelValues, doSpaceBetweenBarChart, barChartColor, cdfLabel];

     (*label="X: Number of arrivals when average rate of arrival is λ";*)
     Row[{pdf, cdf}]
    }]
  )
]
```

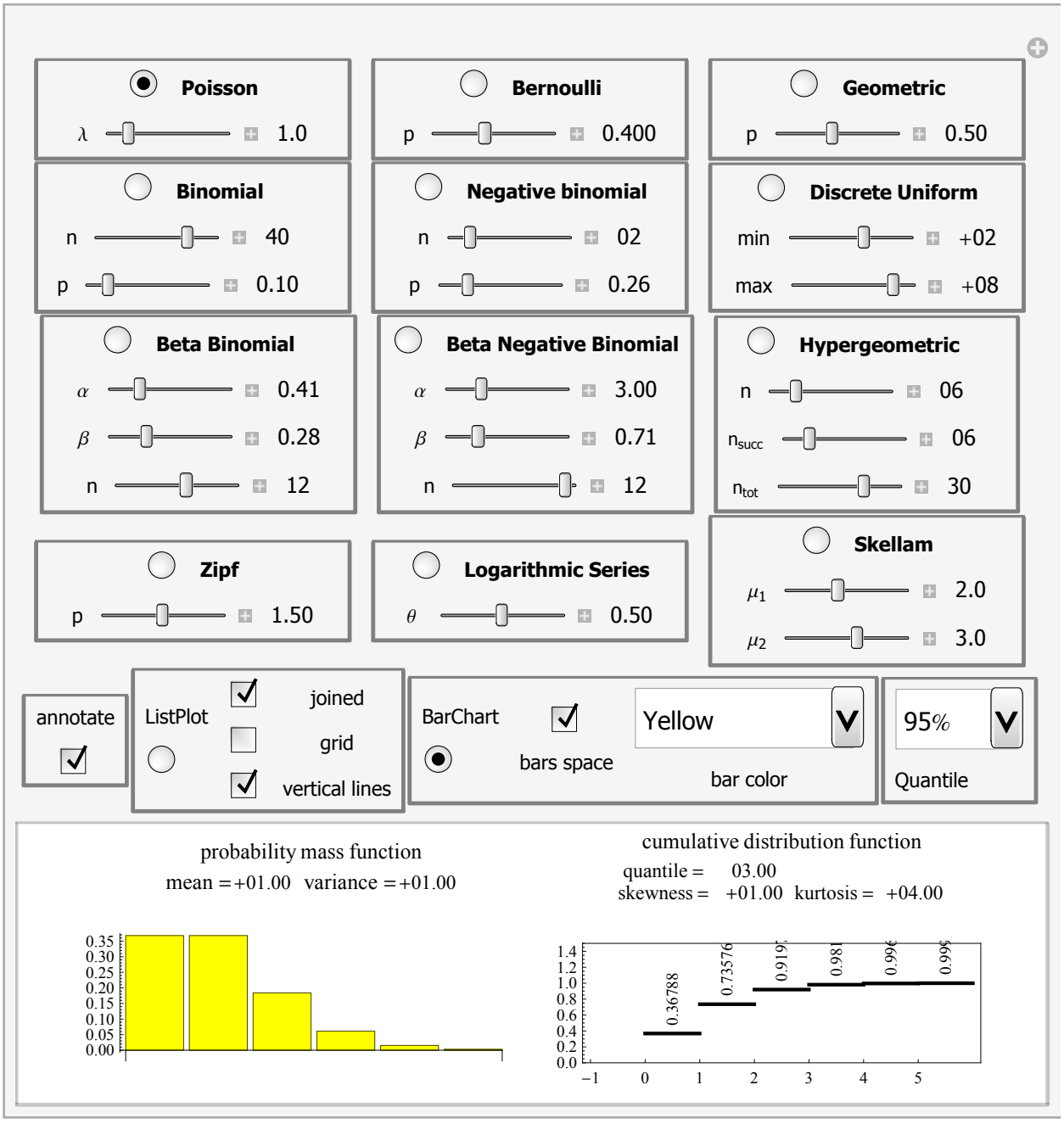


Caption

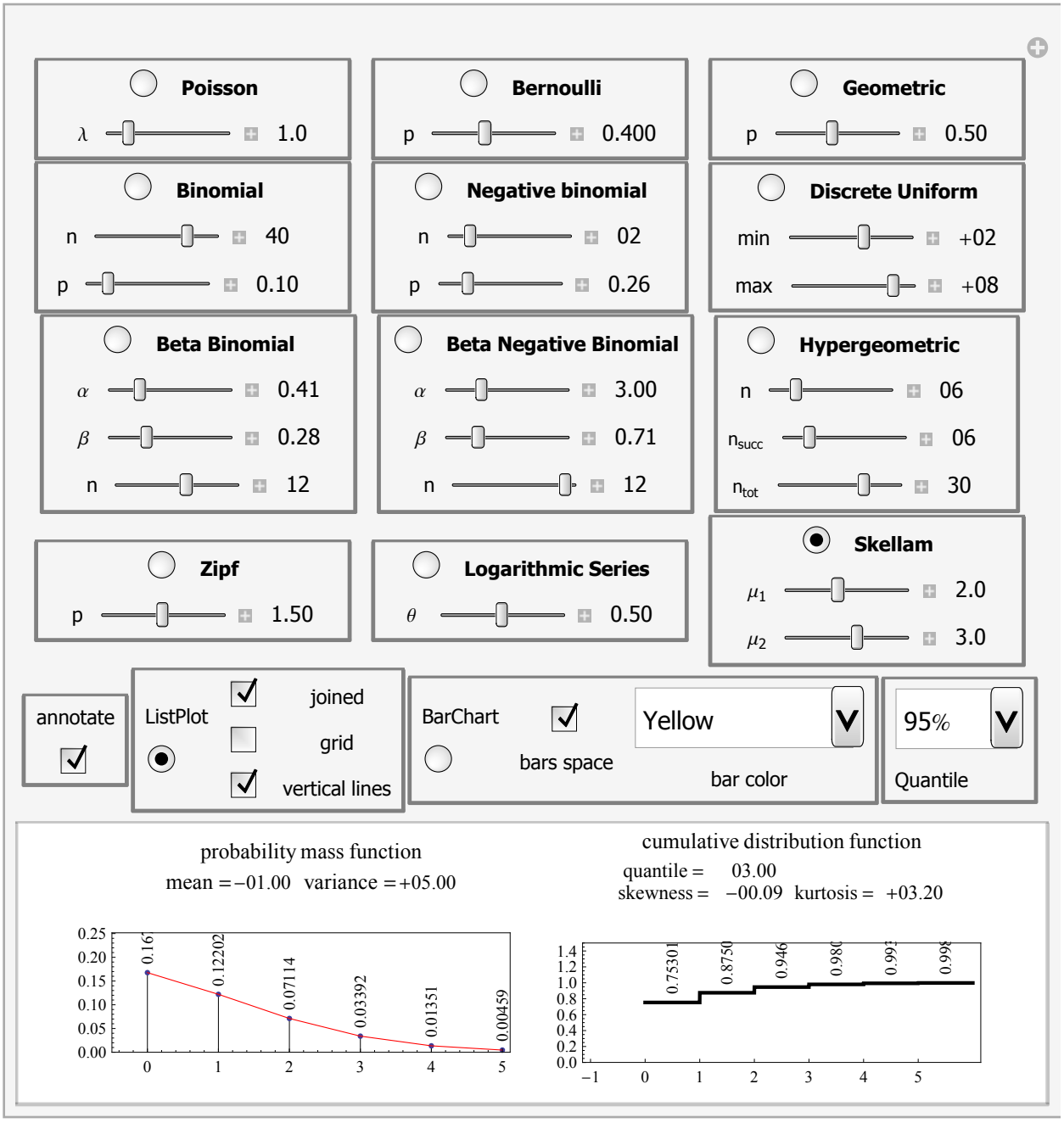
This demonstration illustrates the use of many of of the discrete distributions in *Mathematica* 8.

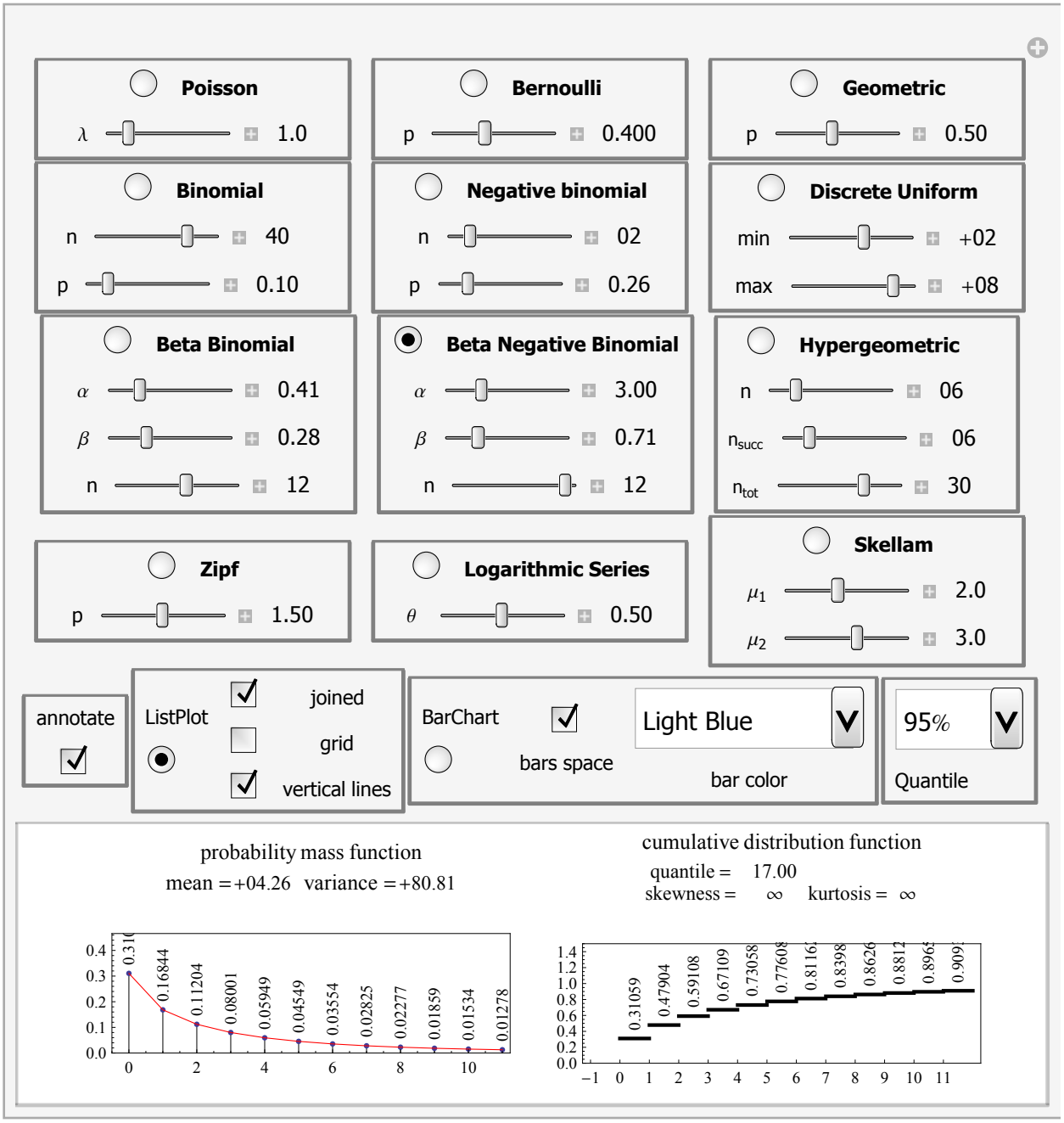
Select the distribution from the display, then select the values of the parameters by using the sliders. The PMF (probability mass function) and the CDF (cumulative distribution function) are displayed. When you move the mouse over each bar in the PMF plot (when in bar chart mode), then the height of the bar will be displayed automatically.

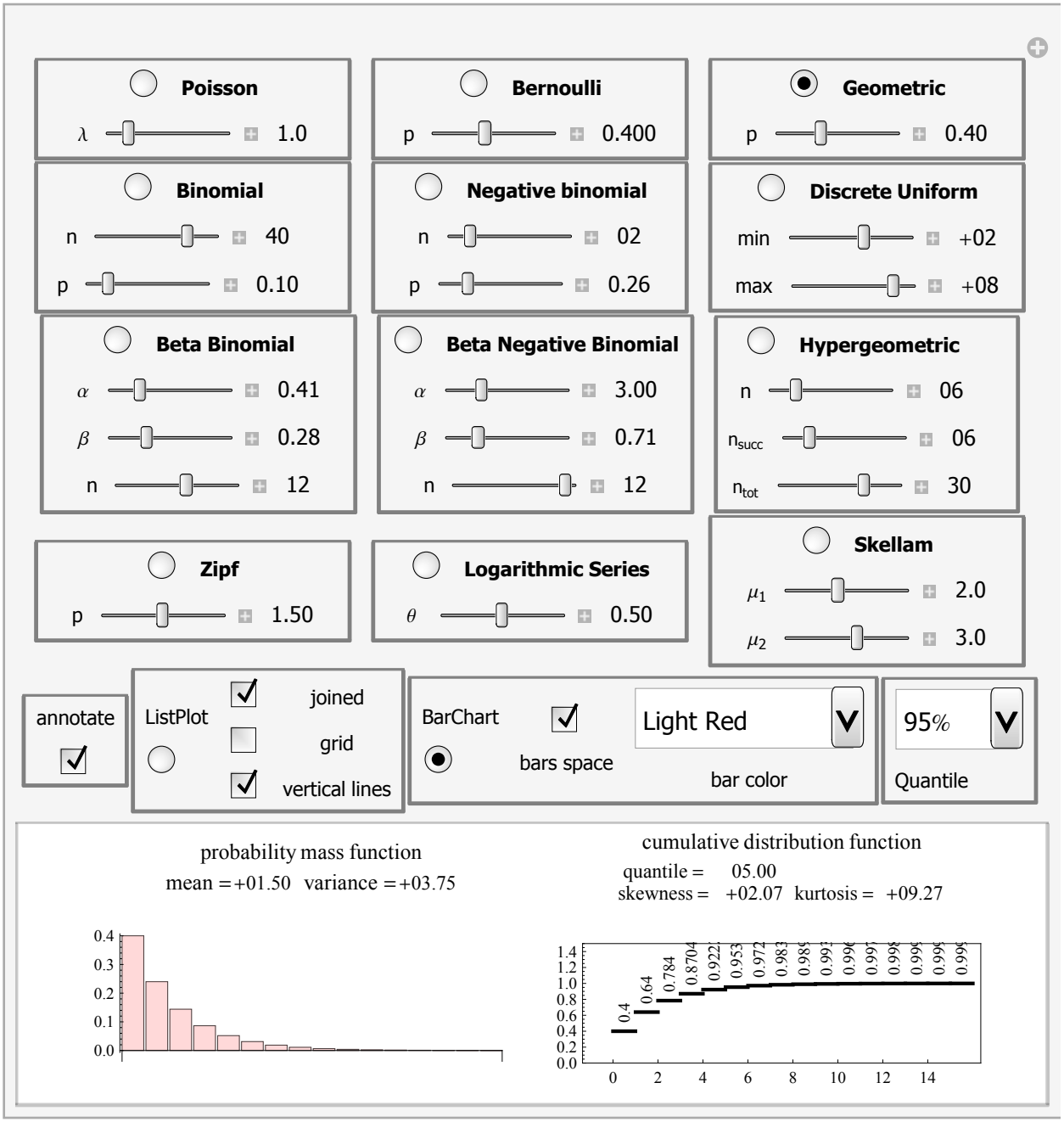
Thumbnail



Snapshots







Details (optional)

Control Suggestions (optional)

- Resize Images
- Rotate and Zoom in 3D
- Drag Locators

- [ ] Create and Delete Locators
- [ ] Slider Zoom
- [ ] Gamepad Controls
- [ ] Automatic Animation
- [ ] Bookmark Animation

Search Terms (optional)

Bernoulli
Binomial
Geometric
NegativeBinomial
HyperGeometric
Poisson
BetaBinomial
BetaNegativeBinomial
LogSeries
DiscreteUniform
zipfDistribution
skellam distribution
Quantile
Discrete distribution

Related Links (optional)

Discrete Distributions

Authoring Information

Contributed by: Nasser M. Abbasi