

How Many Draws to Get to 0.1%?

A Monte Carlo Simulator for Hand-Matching Resistors

by Warren Young, tangentsoft.net

Initialization

Pull in the packages we need:

```
In[1]:= Needs["BarCharts`"];
```

Declare the nominal resistor value, in ohms. The code below is independent of this value, so we just use $1\ \Omega$ to keep things simple. You could change it if you need the saved data set to have some particular scale.

```
In[2]:= nominal = 1;
```

Declare our simulated resistors' specified percent tolerance, as a fraction.

```
In[3]:= mfgPctTolerance =  $\frac{1}{100}$ ;
```

The percent tolerance desired from hand-matching.

```
In[4]:= matchPctTolerance =  $\frac{0.1}{100}$ ;
```

The maximum difference from the nominal value, in ohms, that a resistor is allowed to be and still be in spec

```
In[5]:= maxSpecError = nominal  $\times$  mfgPctTolerance;
```

The minimum and maximum in-tolerance values. These are just for convenience, based on the above values.

```
In[6]:= minInTolValue = nominal - maxSpecError; maxInTolValue = nominal + maxSpecError;
```

Calculate the variance over which to pick random resistor values. The fudge factor was chosen to simulate a "6 sigma" manufacturing process, which gives only a handful of bad values per million units. Increasing the fudge factor simulates a less precise manufacturing process. The actual error for the sample is calculated below to help you adjust this value.

```
In[7]:= var = 0.22 maxSpecError
```

```
Out[7]= 0.0022
```

Number of resistors to generate.

```
In[8]:= trials =  $10^6$ ;
```

Run the Manufacturing and Hand-Matching Simulators

Create the resistor sample set.

```
In[9]:= sample = RandomReal[NormalDistribution[nominal, var], trials];
```

Scan along data sample until we find a pair that are within the desired tolerance. Then, start scanning again from the next element past the set containing the previous match. This method ensures that we never reuse a resistor value, so we get the most out of the entropy we've created. We only do this for a fraction of the sample set...just enough to get a representative data set. You can calculate longer, but it probably won't change the results enough to worry about.

```
In[10]:= tries = {};
For[setStart = 1; setEnd = 2, setEnd <  $\frac{\text{Length}[\text{sample}]}{10}$ , ++setEnd,
  For[i = setStart, i < setEnd && setStart < setEnd, ++i,
    If[matchPctTolerance >
      Abs[sample[[i]] -  $\frac{\text{sample}[[\text{setEnd}]]}{\text{sample}[[i]]}$ ],
      AppendTo[tries, setEnd - setStart + 1];
      ++setEnd; setStart = setEnd
    ]
  ]
]
```

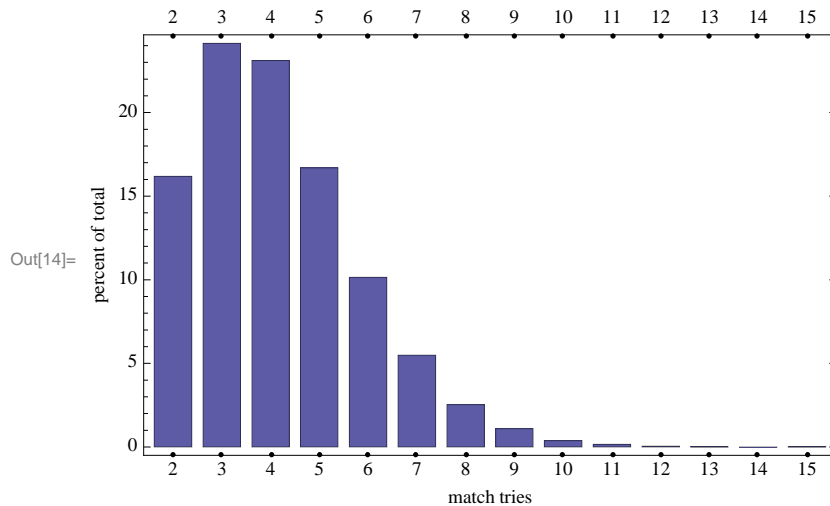
Graph the Results

Summarize the match counts into unit bins starting from 2, the smallest possible match count. Then, rescale the match counts so that the largest value is its percentage of the total, and graph the results.

```
In[12]:= matches = BinCounts[tries, {2, Max[tries], 1}];
```

```
In[13]:= matches =  $\frac{\text{matches}}{\text{Total}[\text{matches}]} \times 100;$ 
```

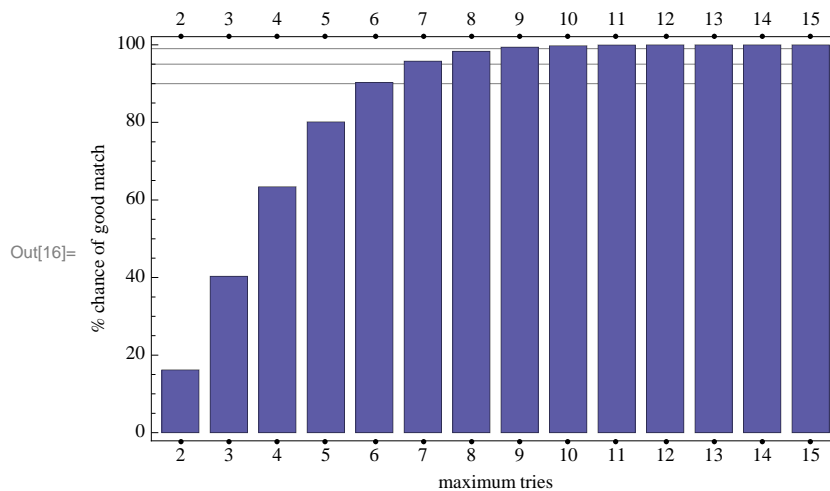
```
In[14]:= BarChart[matches, BarLabels → Range[2, 100],
  Frame → True, FrameLabel → {"match tries", "percent of total"}]
```



Calculate cumulative probabilities of getting a match in N or less tries, and graph the result. The lines show where the probabilities of a good match are greater than 90%, 95%, and 99%.

```
In[15]:= For[i = 1; probaccum = {}, i <= Length[matches], ++i,
  AppendTo[probaccum, Total[Take[matches, i]]]
]
```

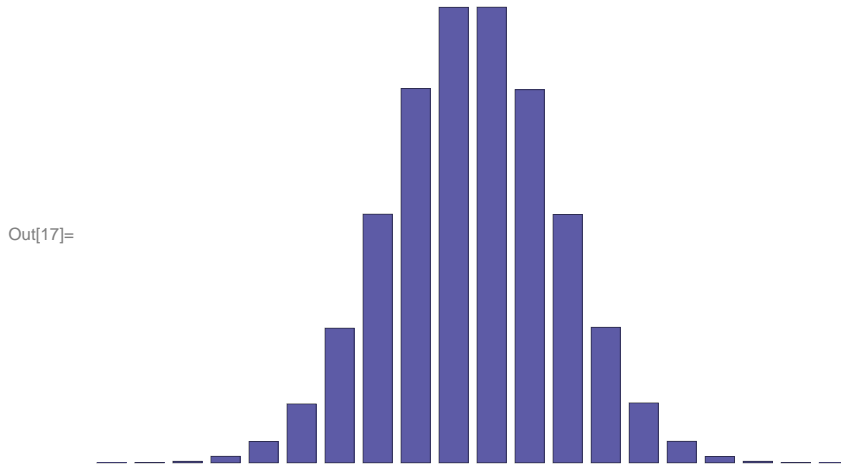
```
In[16]:= BarChart[probaccum, BarLabels → Range[2, 100], Frame → True,
  GridLines → {{}, {90, 95, 99}}, FrameLabel → {"maximum tries", "% chance of good match"}]
```



Data Quality Checking

Graph the data frequency as a visual check that the above parameters are generating a reasonable data set. Also, it's pretty.

```
In[17]:= BarChart[BinCounts[sample, {minInTolValue, maxInTolValue,  $\frac{\text{maxSpecError}}{10}$ }], Axes → False]
```



Check how many outliers we have as a further check on the data's quality. The first result is the number of below-spec resistors, the second is the number of those above spec. Six sigma means 3.4 outliers per million, but as long as both values are in the single digits, it's close enough. I don't like these values to get too low: if it says 0, how can you tell how much better than six sigma you are?

```
In[18]:= {Length[Select[sample - minInTolValue, Negative]],  
          Length[Select[sample - maxInTolValue, Positive]]}
```

```
Out[18]= {5, 1}
```