Positional Tolerances

Howard Gibson C.E.T., hgibson@eol.ca

2018/12/26

1 Introduction

In Geometric Dimensioning and Tolerancing (GD&T), holes are located using positional tolerances. It is claimed that these positional tolerances allow 50% more variation compared to traditional ± tolerances. These extra holes all are functional. This is true if fabricated holes are scattered evenly around the nominal position. It is not true, otherwise.

Dimension units are not being shown on this document. It has been assumed that they are millimetres, but it really does not matter to the discussion. On engineering drawings, the units are declared somewhere on the title-block, and they are not indicated on the drawing itself.

2 Tolerance Specification

Figure 1 shows the areas of a ± tolerance and an equivalent GD&T style positional tolerance. For most holes, the allowable error is a radial displacement. The ± tolerance defines a square that does not exceed this. The GD&T positional tolerance defines the total functional area.

$$TOL = \frac{L_{AB}}{2}$$

$$POS = L_{AC}$$

$$\text{From Pythagoras} \ldots L_{AC} = \sqrt{2L_{AB}^2} = 1.41 L_{AB}$$

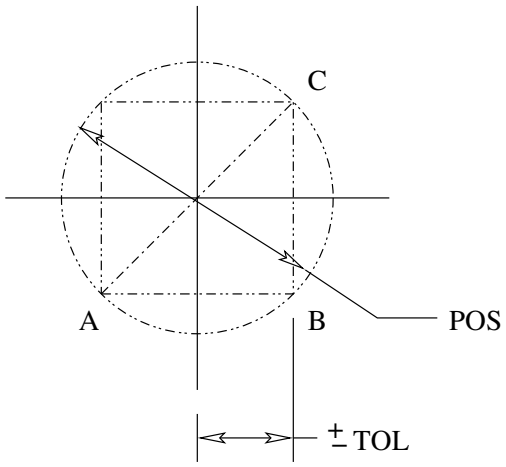

Figure 1: Allowables

$$\text{Area } \pm\text{tolerance: } A_{pm} = L_{AB}^2$$

$$\text{Area positional: } A_{pos} = \frac{\pi L_{AC}^2}{4} = \frac{3.14 \times (1.41 L_{AB})^2}{4} = 1.57 L_{AB}^2$$

So far, so good. The acceptable error for locating most holes is a radius. The GD&T positional tolerance defines this radius. It provides 50% more allowance, all of which is functional.

The problem with this is that as-measured dimensions do not scatter evenly. Generally, they cluster about the nominal position in something like Normal[1] distribution. It is implied that the tolerances called up on drawings represent the $\pm 3\sigma$ interval on the Normal curve.[2] This is not true.

Random scatter is a function of the manufacturing process. Good drafting practice is to specify tolerances required to make the part work. These should be as loose as possible, and it is desirable that they be well outside 3σ of the process. Standard deviation comes from the QC inspector, not the designer.

[1]Normal distribution is discussed in the Appendix .

[2]I have been preparing drawings for over thirty years. I don't set positional tolerances to 3σ. I want 3σ to be well within my positional tolerances.

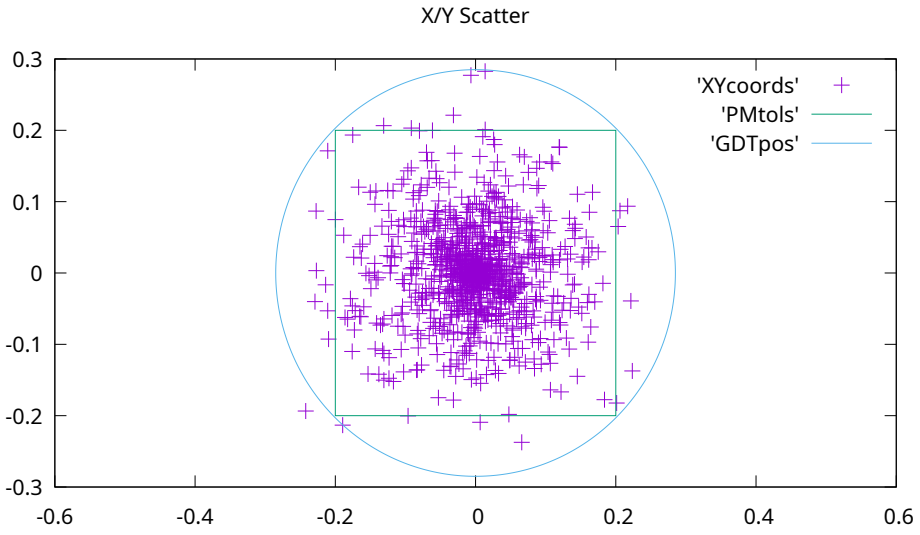

Figure 2: Normal Scattered Data

3 Normal Distribution

For the purposes of analysis, it is assumed that the specified positional tolerance is equal to 3σ. A pattern of holes is randomly generated in a Normal distribution pattern. Each hole position is worked out as a polar coordinate. The angle is randomly generated. The radii are worked out randomly with Normal distribution. The polar coordinates are converted to X/Y. Normal distribution and the Octave code are shown in the appendix. Here are the assumptions...

$\pm$ tolerance	0.2
3σ	0.283
Data points	1000

The positional tolerance has been calculated as $\varnothing 0.570$. On Figure 2 , there are 24 holes outside the $\pm$ tolerance range, out of 1000 total.[3]

[3]This has been run multiple times. For the Normal distribution, we usually get ten to twenty holes outside the specified $\pm$Tolerance.

3.1 Sanity Check

We are generating random data sets, which are supposed to be or to approximate Normal curves. Do they?

We are evaluating the Normally scattered data in Figure 2 .

1. The first we can do is examine Figure 2 and look for holes located outside 3σ. 3σ has been arbitrarily set to GD&T positional tolerance, shown as the circle on Figure 2 . 99.7% of the data should be inside the circle, and 0.3% should be outside it. By observation, on most runs of this program, we see that holes are outside the 3σ circle.

2. We can work out standard deviation for the data. Since I have generated the data from 3σ, it is obvious what σ ought to be. When we work this out for our Normal data set, we get σ=0.097 . This ought to be 1/3 of the 3σ value I punched in, 0.283. This is about right.

3. With Normal distribution, 68% of the radii should fall inside σ. 99.5% of the radii should fall inside 2σ. We can calculate σ from the assigned 3σ, and verify this. It turns out that there are 670 holes located inside σ. There are 952 holes inside 2σ. There are 998 holes inside 3σ. This all is out of 1000 holes.

This is working!

4 Exponent Distribution

The Normal distribution shown above is correct, but difficult to model on a slow computer, or with a spreadsheet. A quick and dirty way to generate random numbers weighted towards zero, is to generate a random number and take it to some exponent.

We have three columns, Normal, Match 1σ, and Match 2σ. We want to match values from authentic Normal data. Exponent values were punched in and tested. As can be seen from Figure 3 and Figure 4 , the scattered data do not look like Normal scattered data. Exponents can be selected which match certain parts of Normal distribution.

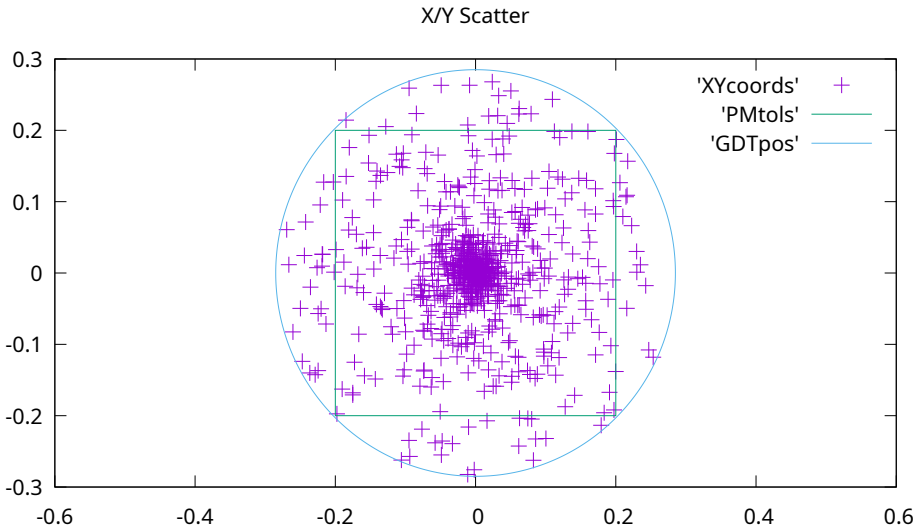

Figure 3: Exponent Distribution $R^{2.8}$

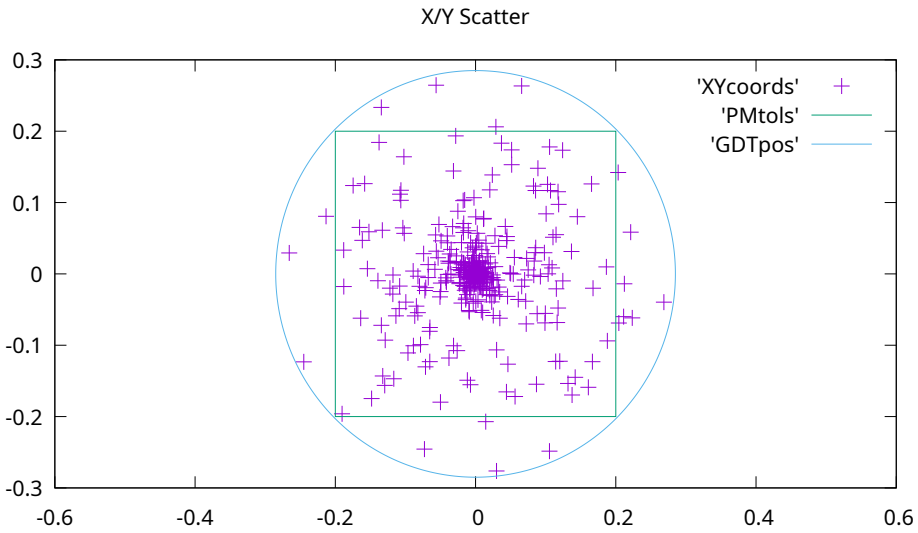

Figure 4: Exponent Distribution R^{9}

	Normal	Match 1σ	Match 2σ
Exponent		$R^{2.8}$	R^9
$\pm$ tolerance	0.2	0.2	0.2
3σ	0.283	0.283	0.283
random sample size	1000	1000	1000
positional tolerance	$\varnothing$0.570	$\varnothing$0.570	$\varnothing$0.570
holes outside $\pm$ tolerance	24	73	18
mean X	0.001672	0.000587	-0.000065
mean Y	0.001601	0.000587	-0.000035
σ from data	0.097	0.106	0.063
holes inside 1σ	670	699	882
holes inside 2σ	952	877	962
holes inside 3σ	998	998	1000

For the Exponent data, the holes inside 3σ sometimes work out to less than 1000. This ought to not happen. The problem seems to be a rounding error somewhere.

5 Conclusions

If we assume that the GD&T positional tolerances approximate the fabricator's 3σ,[4] and that the fabricated holes scatter normally, it is unlikely that more than 2% of holes will be outside the $\pm$ tolerance zone.

The exponent distribution does a poor job of approximating Normal distribution. This might be good enough for a spreadsheet. If you are writing code on a reasonably fast computer, proper Normal distribution is manageable.

6 Why Positional Tolerances?

Assume for the moment that the positional tolerance called up is set to the process 3σ. It can be seen from Figure 2 that the $\pm$ tolerance will capture around 98% of the correctly located holes. It is good design practice to ensure that fabrication capability is well within the required tolerances, when ensures that the $\pm$ tolerances capture even more valid data. This is not a good case for positional tolerancing.

[4] As noted previously, there is no reason why we should.

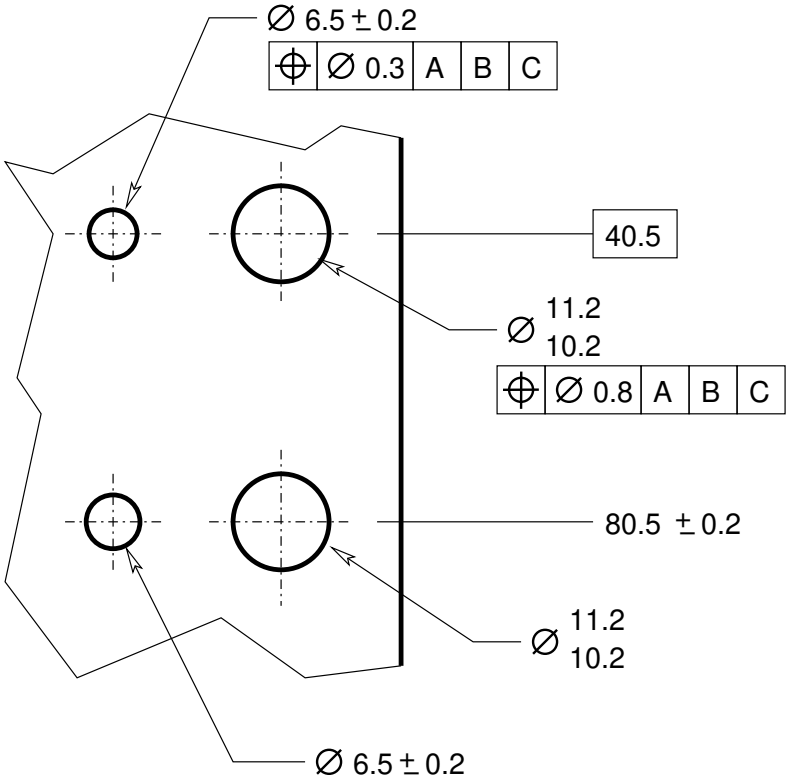

Figure 5: Dimensions to Holes

There is no reason why the distribution of holes should be Normal or even exponent distribution. Scatter may be caused by errors in the gears and encoders of a CNC machine. It may be caused by clearance of bushings in a drill jig. It may be caused by someone mis-reading prints.

Figure 5 shows holes located by ± tolerances, and by positional tolerances. As shown, the 11.2/1.2 hole is located within a 0.8 diameter. On the ± location (80.5), the tolerance must default to the more accurate 6.5 hole. A positional tolerance is part of the hole specification.

A About This Document

This document is written in LaTeX. The data is generated randomly by programs written in GNU Octave, listed in the appendix. The exponents for the exponential distribution were selected by the author after much testing. The scatter diagrams are generated by GNU gnuplot. The final document is assembled and generated with a GNU Makefile.

B Statistics

B.1 Normal Distribution

From Spiegel[2], page 150, we get the equation of a Normal distribution curve...

$$Y = \frac{1}{\sigma\sqrt{2\pi}} \times e^{-\frac{1}{2}(X-\mu)^2/\sigma^2}, \tag{1}$$

...where μ is the mean, and σ is the standard deviation.

B.2 Standard Deviation

From Spiegel[2], page 88,

$$\sigma = \sqrt{\frac{\sum_{j=1}^{N}\left(R_j - \bar{R}\right)^2}{N}}, \tag{2}$$

where N is the number of holes.

I am managing the hole positions as X/Y coordinates. I am converting these to polar coordinates, and I am calculating standard deviation on the radius.

B.3 Exponent Distribution

I cannot find this written up anywhere. A quick and dirty way to generate Normal-like distribution is to generate random numbers between zero and one, and take them to some exponent. The exponent values cluster towards zero. As can be seen by comparing the Normal curve (Figure 2) with Figure 3 and Figure 4), the exponent curves are not that good a model. If you are interested in how many values are out close to the positional tolerance circle, exponent distribution provides a conservative result.

The Normal data is generated by randomly generating numbers along the horizontal and vertical axes, and then accepting anything that falls below the Normal curve. This does not work on a spreadsheet.

C Octave Code

The following programs are executed by the makefile. The output is redirected to files that are read by the LaTeX code.

C.1 Copyright

The author is Howard Gibson, hgibson@eol.ca.

C.2 normal.m

```
#!/usr/bin/octave-cli -qf
# File:      normal.m
# Author:    Howard Gibson
# Date:      2017/11/28
# Project:   Positional Tolerance
# Language:  GNU Octave
#
# Generate a random, normally distributed set of hole positions.
#
# Syntax:  normal.m <PMtol> <3-Sigma> <Size> <Filename>

global x = 1;
global y = 2;
global deg = pi/180;

function Return = roundf(Num, Dec)
  Roundable = Num*10^Dec;
  Rounded = round(Roundable);
  Return = Rounded/10^Dec;
endfunction

function [xv,yv] = polar2XY(a, r)
  xv = r*cos(a);
  yv = r*sin(a);
endfunction

function height = norm(Xval, Mean, Sigma3)
  Sigma = Sigma3/3;
  height = 1/(Sigma*sqrt(2*pi)) * exp(-1/2*(Xval-Mean)^2/Sigma^2);
endfunction

function ScatteredDims = MakeNormal(Sigma3, Size)
  global x y;
  SafetyFactor = 1.5;
  ScatteredDims = cell(Size, 2);
  MaxRadius = SafetyFactor*2*Sigma3;
```

```
  MaxHeight = SafetyFactor*norm(0, 0, Sigma3);
  n = loops =1;
  while n<=Size
    Angle = 2*pi*rand();
    Radius = MaxRadius*rand()-MaxRadius/2;
    Height = MaxHeight*rand();
    if Height < norm(Radius, 0, Sigma3)
      [Xerr, Yerr] = polar2XY(Angle,Radius);
      ScatteredDims(n,x) = Xerr;
      ScatteredDims(n,y) = Yerr;
      n = n+1;
    endif
    loops = loops+1;
  endwhile
  printf("It took %i loops to generate %i data points.\n", loops, Size);
endfunction

if 4!=length(argv())
  printf("Usage:  %s <PMtol> <3-Sigma> <Size> <Filename>\n",
          program_name());
else
  global x y deg;
  PlusMinus = str2num(argv(){1});
  Position = roundf(2*sqrt(2)*PlusMinus,2);
  Sigma3 = str2num(argv(){2});
  Size = str2num(argv(){3});
  Filename = argv(){4};
  ScatteredDims = MakeNormal(Sigma3, Size);
  save (Filename, "ScatteredDims");
endif
```

C.3 sigma.m

```
#!/usr/bin/octave-cli -qf
# File:      sigma.m
# Author:    Howard Gibson
# Date:      2017/11/18
# Project:   Positional Tolerance
```

```
# Language:  GNU Octave
#
# Work out standard deviation for data file.
#
# Syntax:  sigma.m <Data file>

global x = 1;
global y = 2;

function [ang, rad] = xy2Polar(xv, yv)
  rad = sqrt(xv^2+yv^2);
  if xv>0 && yv<0
    ang = 2*pi+asin(yv/rad);
  elseif xv>0
    ang = asin(yv/rad);
  elseif xv<0
    ang = pi-asin(yv/rad);
  else
    ang = 100; # Stupid result.  This should not happen.
  endif
endfunction

function [xm, ym, rm] = mean(ScatteredDims, Size)
  global x y;
  Xtot = Ytot = Rtot= 0;
  for n=1:Size
    Xtot = ScatteredDims{n,x}+Xtot;
    Ytot = ScatteredDims{n,y}+Ytot;
  endfor
  xm = Xtot/Size;
  ym = Ytot/Size;
  for n=1:Size
    [Ang, Rad] = xy2Polar(ScatteredDims{n,x}-xm, ScatteredDims{n,y}-ym);
    Rtot = Rtot+Rad;
  endfor
  rm = Rtot/Size;
endfunction
```

```
if 1!=length(argv())
  printf("Usage:  %s <Data file>\n", program_name());
else
  Filename = argv(){1};
  load (Filename, "ScatteredDims");
  Size = rows(ScatteredDims);
  RadTot = 0;
  [Xmean, Ymean, Rmeanp] = mean(ScatteredDims, Size);
  for n=1:Size
    Xval = ScatteredDims{n,x}-Xmean;
    Yval = ScatteredDims{n,y}-Ymean;
    RadVal = sqrt(Xval^2+Yval^2);
    RadTot = RadTot + RadVal^2;
  endfor
  sigma = sqrt(RadTot/Size);
  printf("%6.3f", sigma);
endif
```

C.4 exponent.m

```
#!/usr/bin/octave-cli -qf
# File:      exponent.m
# Author:    Howard Gibson
# Date:      2017/12/02
# Project:   Positional Tolerance
# Language:  GNU Octave
#
# Generate a random, square root set of hole positions.
#
# Syntax:  exponent.m <exp> <PMtol> <3-Sigma> <Size> <Filename>

global x = 1;
global y = 2;
global deg = pi/180;

function Return = roundf(Num, Dec)
  Roundable = Num*10^Dec;
  Rounded = round(Roundable);
```

```
  Return = Rounded/10^Dec;
endfunction

function [xv,yv] = polar2XY(a, r)
  xv = r*cos(a);
  yv = r*sin(a);
endfunction

function ScatteredDims = rootMeanSquare(Exponent, Sigma3, Size)
  global x y;
  ScatteredDims = cell(Size, 2);
  MaxRadius = Sigma3;
  for n = 1:Size
    Angle = 2*pi*rand();
    Radius = MaxRadius*rand()^Exponent;
    [Xerr, Yerr] = polar2XY(Angle,Radius);
    ScatteredDims(n,x) = Xerr;
    ScatteredDims(n,y) = Yerr;
    if Radius>MaxRadius # This should not be happening
      printf("Radius = %5.3f\n", Radius);
    endif
  endfor
endfunction

if 5!=length(argv())
  printf("Usage:  %s <exp> <PMtol> <3-Sigma> <Size> <Filename>\n",
         program_name());
else
  global x y deg;
  Exponent = str2num(argv(){1});
  PlusMinus = str2num(argv(){2});
  Position = roundf(2*sqrt(2)*PlusMinus,2);
  Sigma3 = str2num(argv(){3});
  Size = str2num(argv(){4});
  Filename = argv(){5};
  ScatteredDims = rootMeanSquare(Exponent, Sigma3, Size);
  save (Filename, "ScatteredDims");
endif
```

References

[1] Tolerance Stack-Up Analysis [for Plus and Minus and Geometric Tolerancing] Second Edition *James D. Meadows*

[2] Schaum's Outlines – Statistics Second Edition *Murray R. Spiegel* McGraw Hill