Visualization of 3D Parameter Spaces in 2D: A Novel Method for Data Exploration

Stefan Nikles, Ph.D.
Analog Devices, Inc.

Introduction

Several JMP platforms exist for visualizing parameter spaces: Contour Plot, Overlay Plot, Surface Plot, and Scatterplot 3D, to name a few. These work well when exploring data containing a single dependent variable as a function of 2 independent variables, but it is more difficult when one needs to explore data as a function of 3 independent variables (e.g. a 3-factor DOE). Users must commonly resort to presenting multiple plots for all values of one or more of the parameters, making it difficult to discern important relationships. In many cases however, the primary concern is simply knowing where the dependent variable exceeds a specific threshold. For example, under what conditions does the signal-to-noise ratio exceed unity? Or where in the parameter space is the p-value less than 0.05?

In this paper, a method is described for computing a single contour level from multiple plots and overlaying each into a single plot. This method employs a GUI-enabled JSL script to compute the contours, which are then plotted using the Graphics Scripting feature within the Bivariate Platform. The use of this method was essential in validation of a new measurement technique for our devices.

Background: Methods of Visualizing Multivariate Data

The end goal of any experiment is to understand the relationships between inputs and output so that some predictive theory or fundamental understanding may be developed. Often however, either the true relationship is so complex that it defies any intuitive understanding how the independent variables affect the output, or the underlying relationship is simply unknown. Proper data visualization is a fundamental necessity in these instances in order to guide understanding and further experimentation.

Visualization is straightforward when a data set consists of only one or two independent variables. A simple XY plot or contour plot will suffice. But what about when the data set consists of 3 or more independent variables? How can this data be represented in an intelligible manner, so we may easily draw conclusions from it? Some common approaches:

- **Many graphs for every combination:** Plot dependent variable against only one independent variable. Multiple plots must be used for each combination of additional independent variable values. In JMP, this can be done using Bivariate Plot, or Graphbuilder (Figure 1)

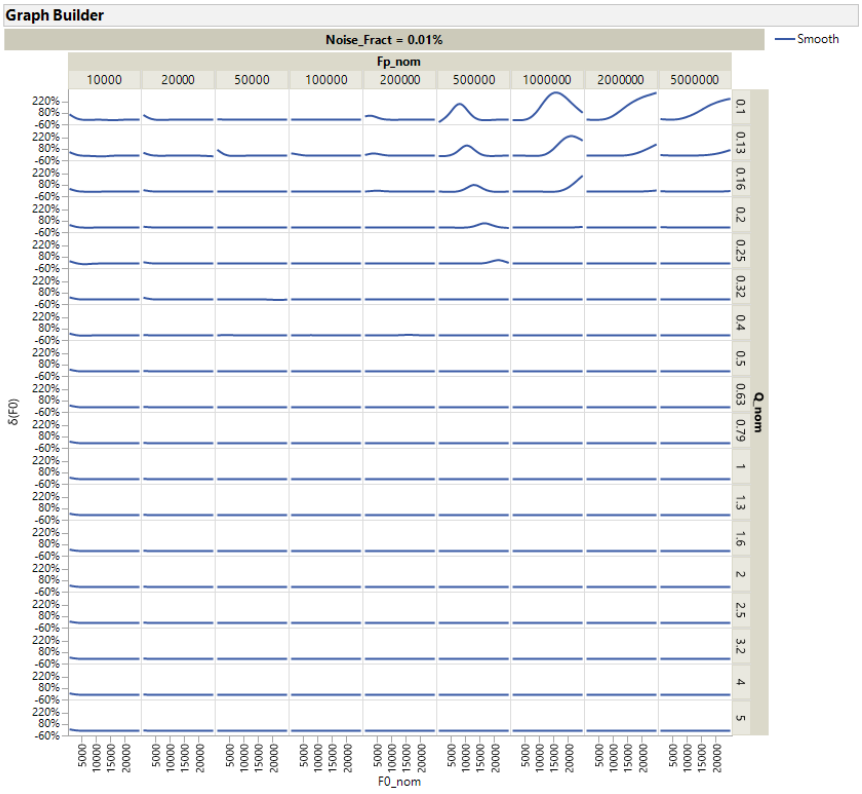

Figure 1: Using Graphbuilder to generate multiple plots of the dependent variable against one independent variable, with one plot for each combination of additional independent variable values.

- **"One graph to rule them all":** Plot dependent variable against only one independent variable but using only a single plot with multiple curves overlaid or different colored points in the plot for different values of additional independent variables. In JMP, this can be done using Bivariate Plot, Overlay Plot (Figure 2), or Graphbuilder

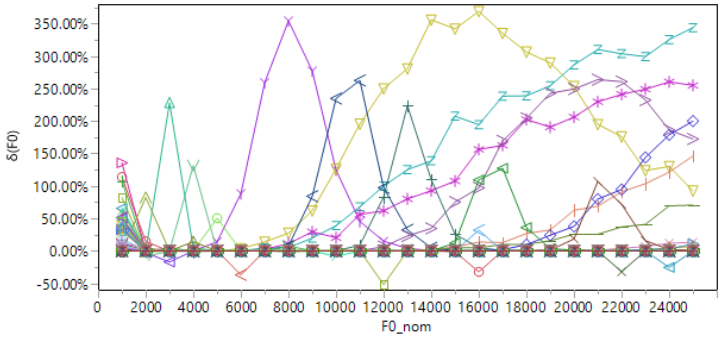

Figure 2: Using a single Overlay Plot to plot the dependent variable against an independent variable. Multiple curves are overlaid in the plot for all different values of additional independent variables.

- **Contour Plots (2D Plots):** Plot dependent variable against two independent variables, with multiple plots for different values of additional independent variables. In JMP, this can be done using Contour Plot, Surface Plot, or Graphbuilder (Figure 3).

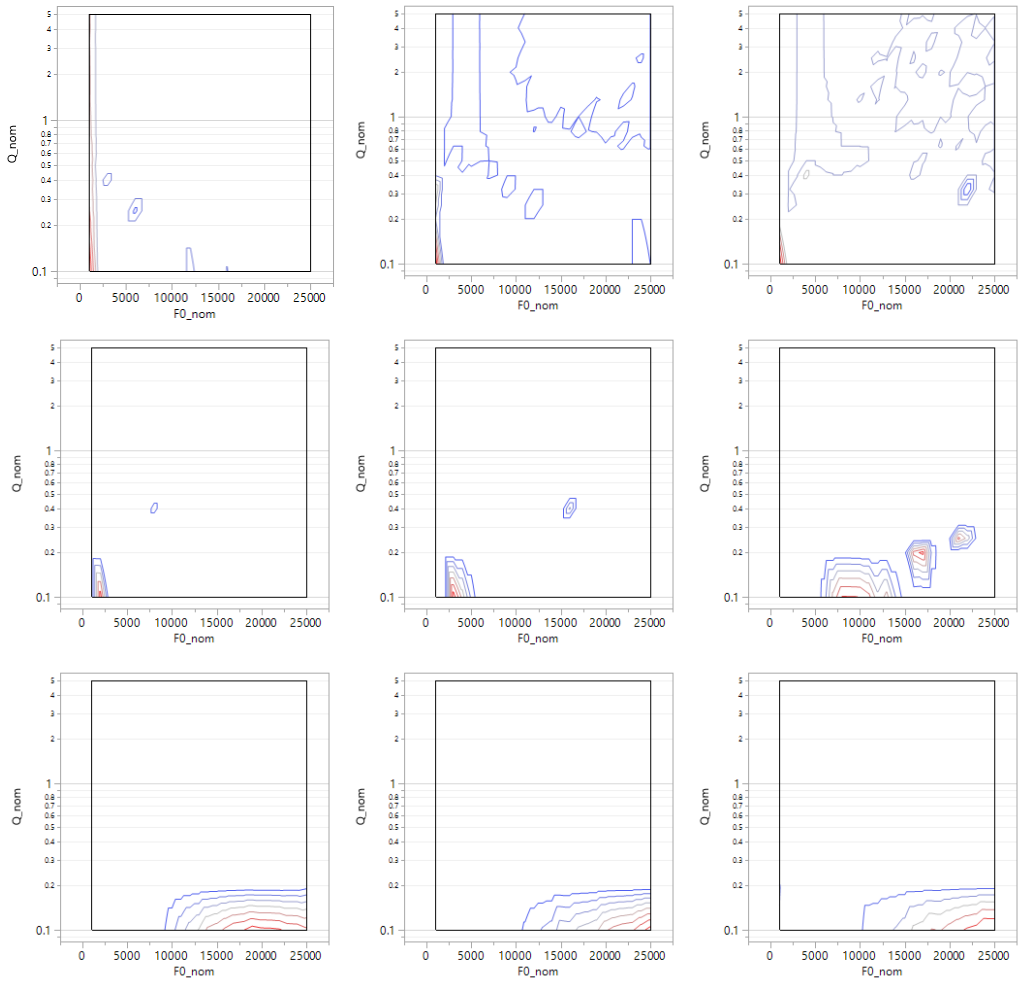

Figure 3: Using the Contour Plot to show the dependent variable against two independent variables. Multiple plots must be used for different values of additional independent variables (Fp_nom: 10kHz – 5MHz).

- **3D Plots:** Plot dependent variable against three independent variables with points colored according to the dependent variable, and as before, multiple plots for different values of additional independent variables. In JMP, this can be done using Scatterplot 3D (Figure 4)

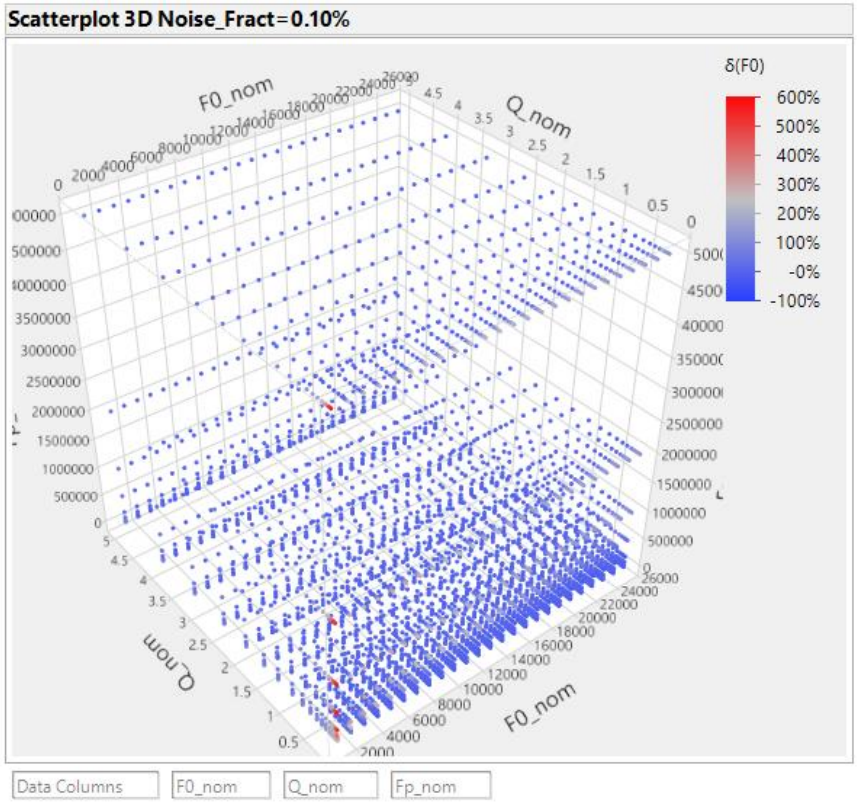

Figure 4: Using Scatterplot 3D to show how the dependent variable varies against 3 independent variables. Here the dependent variable is represented by the point color.

These purpose of showing these figures is to demonstrate how difficult it can be to practically visualize and interpret complex datasets. The common problem is that there are simply too many data to represent at once. Sometimes we may have the luxury of knowing that a particular input has no significance on the output. We may even decide to narrow the focus on a subset of the parameter space where only one or two inputs have an effect. This is unfortunately a luxury we do not always have. How then can multivariate data be presented? Would it be possible to reduce the data in another way, so it could be plotted more effectively?

Motivation: MEMS Accelerometer Production Test

Micromachined (MEMS) accelerometers sense accelerations, and are used widely throughout industry: Automobiles, Electronics, Implantable Biomedical Devices, Navigation, etc. Two important characteristics of any accelerometer are known as F0 (Natural Frequency) and Q (Quality Factor). These define how the device behaves when it is vibrated. It is necessary to measure these parameters on our die during production because they provide information about each device's inherent sensitivity (F0) and integrity (Q). Figure 5 provides images of a typical MEMS accelerometer fabricated at ADI.

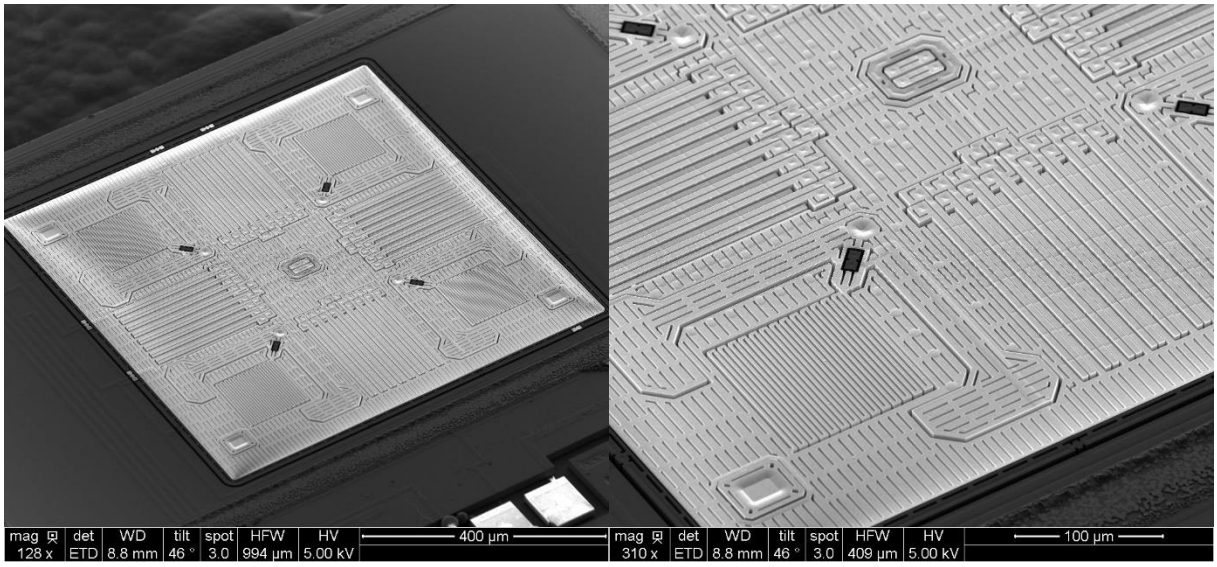

Figure 5: SEM of a MEMS accelerometer (ADXL362)

During a recent development phase for a new accelerometer design at ADI, initial measurements of these parameters did not agree well with theory. Several explanations were proposed and tested, but eventually it became evident that the problem was with the measurement method. In this method, the accelerometer is vibrated electrostatically at several frequencies and the output response is measured. Curve-fitting is then used to extract F0 and Q from this data. This method had agreed well with theory in the past, but not for this new design.

The question asked was "How accurate is our measurement method over a range of F0 and Q values for our designs?" One possibility was that noise in the measurement had a greater impact on the measurement accuracy for some values of F0 and Q than others.

A mathematical model of the accelerometer was created to explore this issue further. This model was defined by nominal values of F0 and Q, as well as possible parasitic losses in the measurement system, defined by a third parameter Fp. The accelerometer response over several frequencies was simulated and random noise was added, whose level was defined by a fourth parameter noise_fract. In total, 16,200 parameter combinations were simulated (Table 1). For each combination, our method was applied to the data to re-compute F0 and Q. Finally, the percentage errors between the computed values and the nominal values, δF0 and δQ, were determined.

Table 1: Parameters used to generate MEM accelerometer model.

Parameter	**Values Simulated**	
F0_nom	1kHz - 25kHz	(25 values)
Q_nom	0.1 – 5.0	(18 values)
Fp_nom	10kHz – 5MHz	(9 values)
noise_fract	0.01% - 10%	(4 values)
Total	16,200 combinations simulated	

It became clear very soon after performing this large set of simulations that navigating the results would be exceptionally difficult. At a minimum there are two dependent variables: δF0 and δQ; and 4 independent variables: F0, Q, Fp, and noise_fract (more if other measurement settings are included). Figure 1 through Figure 4 show the attempts to visualize this data set for just one value of noise_fract. As it can be seen, it is difficult to get a holistic view of the parameter space from these methods.

However, to evaluate the production test, the primary concern is that the measurement error be less than some agreed upon value (e.g. ±5%). It will always be preferred to have a measurement error as low as possible, but the key factor here is simply knowing that the measurement error is within this threshold. A method for doing this will be discussed in the next section.

Method

The Threshold Plot

As mentioned above, suppose we care only about the regions where a variable exceeds a certain threshold? In this case, we can reduce the data set to just two regions: those where the variable meets the threshold criteria and those where it does not.

A "Threshold Plot" shows all the overlapping regions in a 3D parameter space where a certain criterion is met. It is a 2D plot with 2 independent variables plotted on the X and Y axes, and a curve showing the border between locations where the dependent variable is above or below a given threshold. Multiple borders may be overlaid to represent different values of a third independent variable.

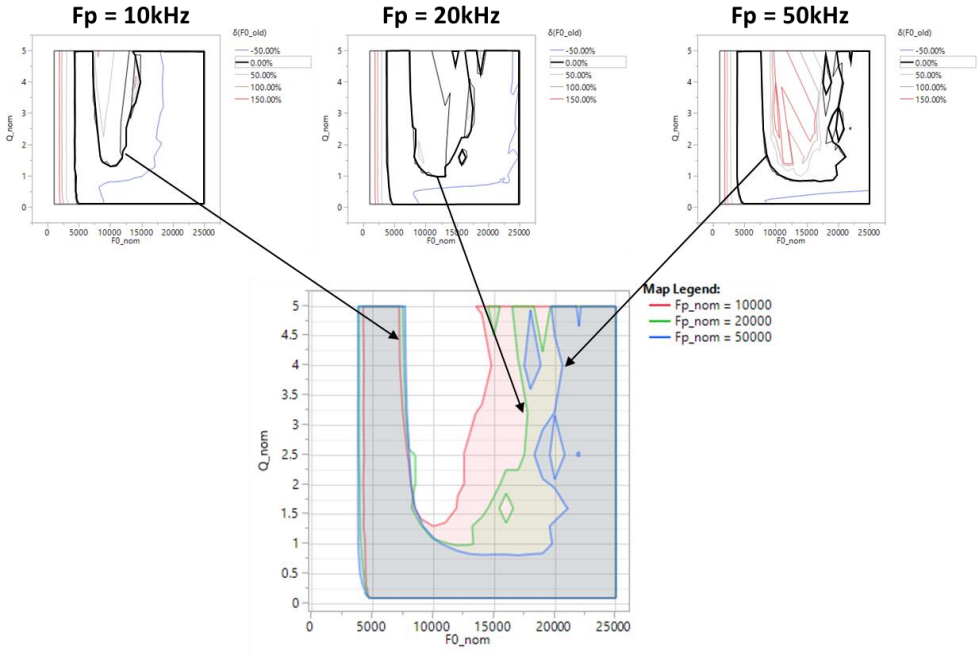

Figure 6: The relationship between contour plots (top) and a threshold plot (bottom). Here, three contour plots are generated, one for each value of an independent variable, Fp. A single contour level is taken from each plot and overlaid in the threshold plot. This shows all the regions through the parameter space where the criteria are met.

Figure 6 represents the relationship between contour and threshold plots using 3 independent variables: F0_nom, Q_nom, and Fp_nom. Three contour plots were generated for each value of Fp_nom, with the X & Y axes representing F0_nom and Q_nom respectively, and the contour levels representing the dependent variable δF0. The threshold plot at the bottom of the figure shows a single contour level from each contour plot (here, δF0 = 0%), merged onto a single plot.

The advantage of the threshold plot is that the key information may be compressed into a single plot without overwhelming the audience. Additionally, the overlapping regions where the criterion is met for all independent variables can be easily observed. This would be very difficult using the visualization methods described in Figure 1 through Figure 4.

Borders.jsl: Operation and Guidelines

A JSL script was created to generate threshold plots in JMP. The script creates an input GUI, computes the borders, and displays the results in a separate window. Steps for operation are simple:

1. Open table containing data to be analyzed.
2. Assign various columns into roles.
3. Set the test columns and limits.
4. Set the test condition.
5. Click ok.

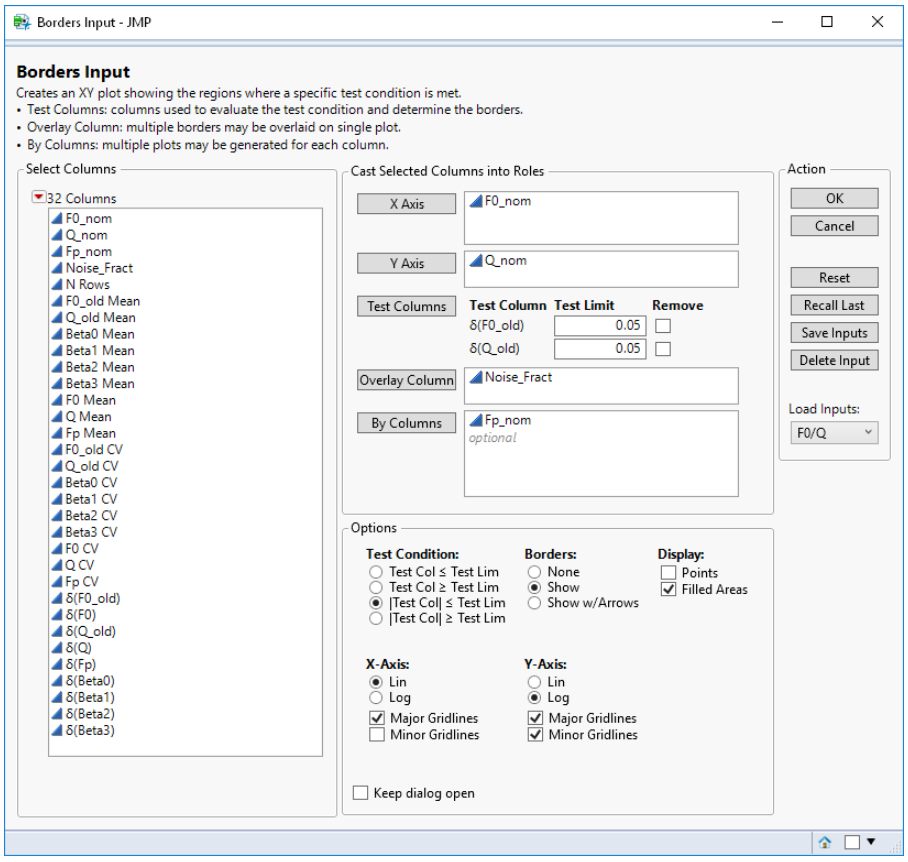

Figure 7: The Borders.jsl input GUI.

Explanations of the various elements of the GUI (shown in Figure 7) are given below:

- **X/Y Axis:** 2 independent variables (must be numeric). These will form the X & Y axes of the Threshold plot.
- **Test Columns** (must be numeric). The border in the plot will be created based on where this column meets the test limit. Separate Threshold plots will be created for each column provided.
- **Overlay Column:** 3rd independent variable (numeric or character). Separate borders will be created in the plot for each value of this variable.
- **By Columns**: Additional plots may be created for each combination of values in the By Columns.
- **Test Condition**: Allows the border to be computed different ways (greater than limit, less than limit, etc).
- **Recall Last**: User can recall last set of analysis inputs
- **Save Inputs /Delete Input /Load Inputs**: Frequently repeated analysis inputs can be managed for later use.

The GUI was intentionally structured to be similar to other common JMP platform GUIs. This aids in ease of use and increases likelihood of adoption. One useful feature added here was the ability to save frequently used input settings using "Save Inputs" and recall them later using "Load Inputs".

Some restrictions and guidelines apply to this script:

- As mentioned above, the X, Y, and Test columns must be numeric, but the Overlay and By columns may be numeric or character type.
- The X, Y, and Overlay columns should have reasonably small number of levels (e.g. <20) to limit run time and plot complexity. For example, in a nutrition study where each subject has a unique weight, these values may be combined in to 5kg groups using a binning formula in JMP (Cols > Utilities > Make Binning Formula).
- All combinations of X & Y values need *not* be present in dataset (these points will just be treated as outside the threshold). For example, in a nutrition study where subjects are organized by age and weight, not every weight may be represented within each age group. Too many missing points will result in a threshold plot that is difficult to read though (see Figure 8). As before, a binning formula may be helpful.
- Multiple values may exist in the table for a single X & Y combination. These will just be averaged together. If this is not desired, the values may be differentiated by adding another column to the "By Column" field.

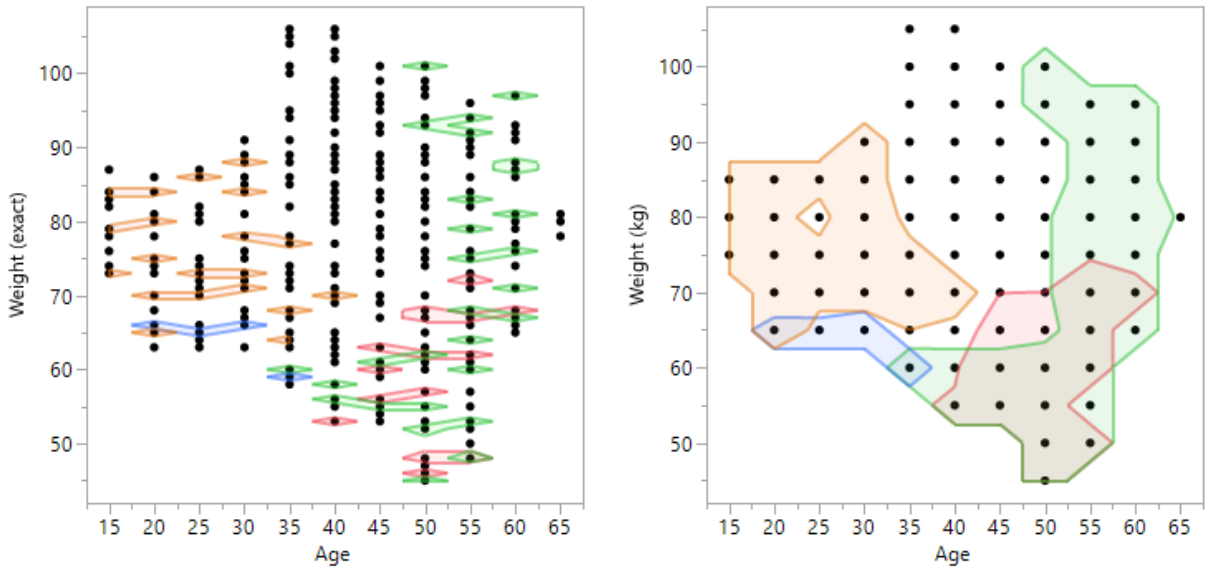

Figure 8: Example of having too many missing data points (Left). This can be resolved by binning the column (Right).

Borders.jsl: How the Code Works

A plot is created for each combination of By column values. Within each plot, a border is created for each value of the Overlay column. These borders are created in 3 passes through the data:

1. First, each element is tested against the threshold criteria (Figure 9A).
2. In the second pass, those elements that are on a region border are identified (Figure 9B).
3. In the third pass, each time a border element is encountered, a point is interpolated between that element and the element outside based on the test limit value (Figure 9C).
    - These interpolated X & Y coordinates are then saved to an array.
    - The next adjacent border element in the counter-clockwise direction is then identified and the next interpolated point is saved to the array.
    - This is repeated until the border loops back on itself.
    - This is continued through the entire data set until all elements have been checked.

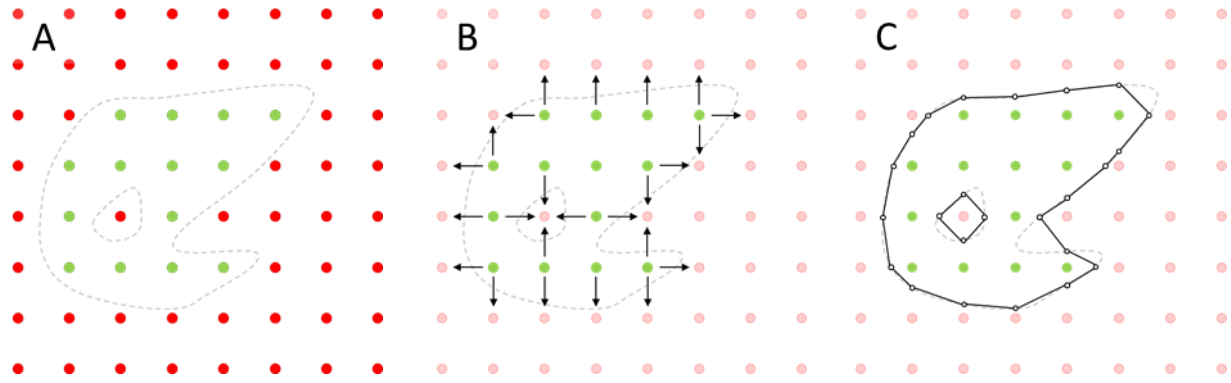

Figure 9: Graphical representation of the steps in Borders.jsl used to create the borders. See text for details.

Once the interpolated points have been saved to an array, the Bivariate Plot platform is used in to create the outputs. A simplified version of the code is shown below:

```
bvp = Bivariate(
    SendToByGroup(Bygroup Default),
    Y(Column(dt, ycol)),
    X(Column(dt, xcol)),
    …additional Dispatch commands…

    Dispatch(
        {},
        "Bivar Plot",
        FrameBox,
        {Add Graphics Script(
            Pen Color(bordercolor);

    //border.arr contains the X&Y coordinates for each line segment of the border
    //border.arr[i][1] = X coordinate for the ith border segment
    //border.arr[i][2] = Y coordinate for the ith border segment
            For(i=1, i<=NItems(border.arr)-1, i++,
                Pen Size(2);
                Transparency(0.5);
                Line(
                    {border.arr[i][1], border.arr[i][2]},
                    {border.arr[i+1][1], border.arr[i+1][2]},
                );
            );              //end of looping through each segment
        )}          //closing Graphics Script
    )
)
```

Here, the array containing all the coordinates to create a border is called “border.arr”. Each border has a separate array. The border is drawn in the bivariate plot using a Dispatch command, containing an Add Graphics Script() command. This command allows the user to execute any JSL code when the plot is created. In this case, a For loop is used to sequentially draw each line segment of the border, referring to border.arr for the coordinates.

This is only a summary explanation of the script operation, but it covers the main operating points. The script may be consulted directly for more detail.

Results

The MEMS accelerometer simulations were performed as described in the earlier section, and Borders.jsl was used to create the relevant threshold plots, shown in Figure 10. Immediately it was obvious where in the parameter space the test method worked and where it did not. For designs having an F0 of ~7kHz and Q of ~1.5, the plots show that the method would have an error of less than ±5%. Unfortunately, the plots also show that the test method would be insufficient for the new accelerometer design (F0 = 19kHz, Q = 0.6).

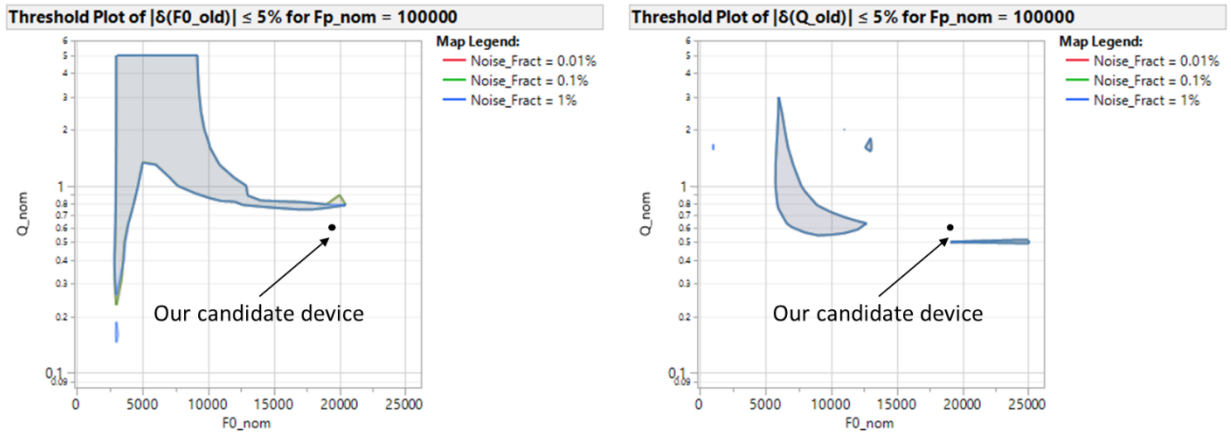

Figure 10: Threshold plots for F0 and Q, showing all regions where the test method has less than 5% error.

Based on this analysis, the method was improved. Parasitic losses inherent to the measurement system were incorporated into the model. These losses had always been present but were not a critical issue for previous accelerometer designs being measured on this particular system.

The method of computing F0 and Q was re-derived after incorporation of the new model. The new method was applied to the same simulated data as before and threshold plots generated (Figure 11). They demonstrate that the method improvement gives us less than 5% error even for Noise_Fract up to 1%. If system noise were reduced further, the method would work over nearly the entire parameter space.

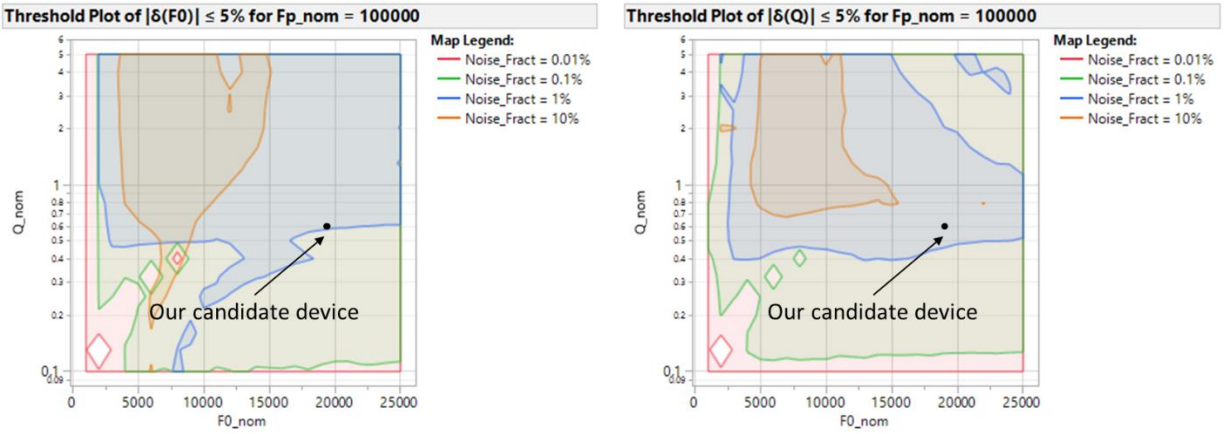

Figure 11: Threshold plots for F0 and Q using the revised measurement method, showing all regions where the test method has less than ±5% error.

These results were crucial to driving the improvement of the old method and adoption of the new. The revised method is now in use on two new accelerometer products at ADI.

Conclusion

In this work, we have introduced a new type of visualization, called the “Threshold Plot”. This allows users to simultaneously view in one 2D plot all regions in a 3D parameter space where the dependent variable or test result meets a given criterion. A GUI-enabled script “Borders.jsl” was created to generate these plots. The script is versatile, user-friendly, and effective. This method offers improvements over existing visualization methods such as Contour Plot or Overlay Plot. We have successfully used this method to identify deficiency and drive improvement of a measurement method used for MEMS accelerometers.