Pyghack 2018

September 29, 2018 – September 30, 2018

Sudesh Sahu
Jatin Mathur
Himanshu Minocha
Amodh Vyas

Problem Statement:

IDOT (Illinois Department of Transportation) has been inspecting bridge elements (e.g. steel column or pile extensions; unpainted steel floor beams) since 1995. Bridge elements are described in IDOT's Bridge Element Inspection Manual. Historical inspection data for each bridge element for each bridge is stored in SQL tables in an Access Database.

IDOT needs to develop deterioration curves based on the historical data it has collected while inspecting bridge elements since 1995. There should be a deterioration curve for each bridge element in each kind of environment (benign, low, moderate, and severe).

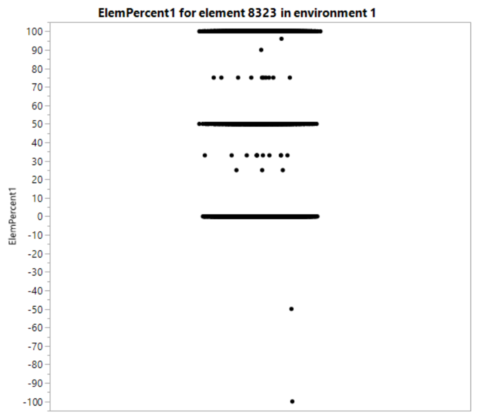
Judging Criteria:

- Which solution generates the most accurate/reasonable deterioration curves for each bridge element for each of the four environments?
- Does the solution have the ability to display the deterioration curves graphically?
- Has the solution analyzed the historical inspection element data to determine what data shall be included to produce reasonable deterioration curves? For example:
    - The element may have been repaired or replaced over the past 20-years which could skew the results.
    - Bad inspection data could skew the results (key typing errors, small sample size, etc.).
    - Etc.

Our Solution

Note: to properly explain our solution and our method, we need to explain some of the problems we ran into. That will be included in the following.

Simply based on how the question is phrased, IDOT made it seem like there should be a relatively clear deterioration curve for each structure in each environment. The question was only what kind of deterioration exists (linear or exponential decay were the most likely candidates) and how that deterioration is affected by different environments. Our team was familiar with JMP, a software tool for statistical analysis, so we loaded the Excel data into JMP and created the following graph:



These are the problems:

1) Some points had invalid entries (such as negative numbers) for the percent of structures
2) We had no way to combine the percent of structures in condition 1, condition 2, condition 3, and condition 4 (lower the better) into one overall, comparable score. The graph just uses “ElemPercent1” (percent of elements in condition 1)
3) Some structures had few overall quantities, meaning they skewed deterioration curves. For example, most instances of the element in the graph shown had a total quantity of two (sometimes there were more, which explains the points in between, but the vast majority of instances of the element were limited to that), so the percent of structures in any condition is either 100%, 50%, or 0%, making it artificially stepwise in decrease.
4) There was nothing existing that could be put on the “x” dimension.

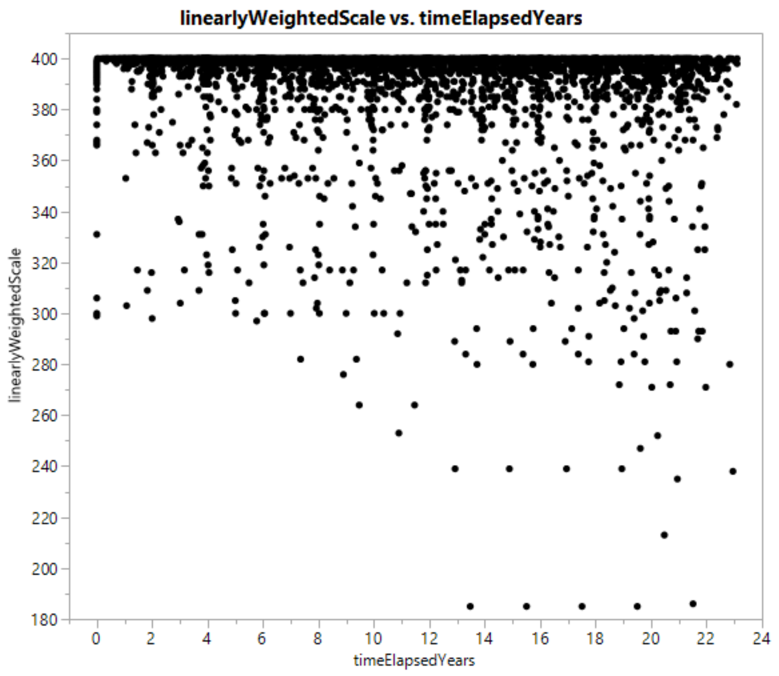
These were the solutions and assumptions:

To address issue 1, we had to filter, find, and exclude those data points. To address issue 2, we defined the following scoring function:

Score = percentCond1 * 4+ percentCond2 * 3 + percentCond3*2 + percentCond4*1

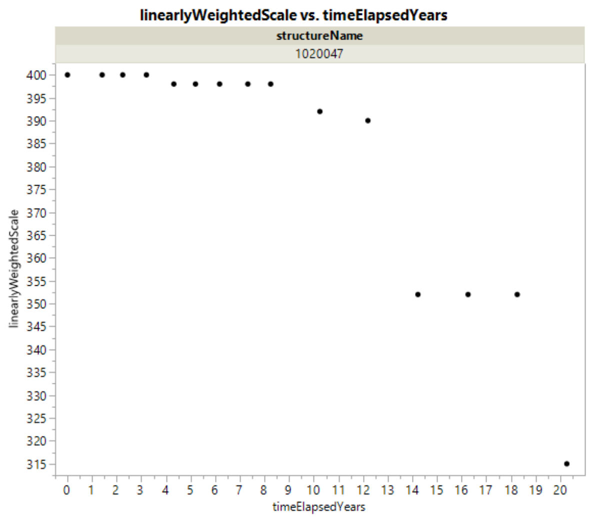
We created this formula on the assumption that condition 1 and condition 2 are as roughly far apart in quality as condition 2 and condition 3. Essentially, we assumed a linear difference in the conditions, so that we could sum them linearly. And the way this formula is created, the lower the score the worse (as that would mean more "percents" are in worse conditions). The baseline score is 400. We confirmed this with our point of contact, Hugh Gallivan, and he agreed that since it wasn't clear how to combine the conditions into a score, this was a good approach. To address issue 3, we ignored those elements altogether, instead filtering and finding data that had at least ten as their total quantity. To address issue 4, we created a new column that formulaically determined how much time elapsed for an element in a given building since that element specifically was first inspected.

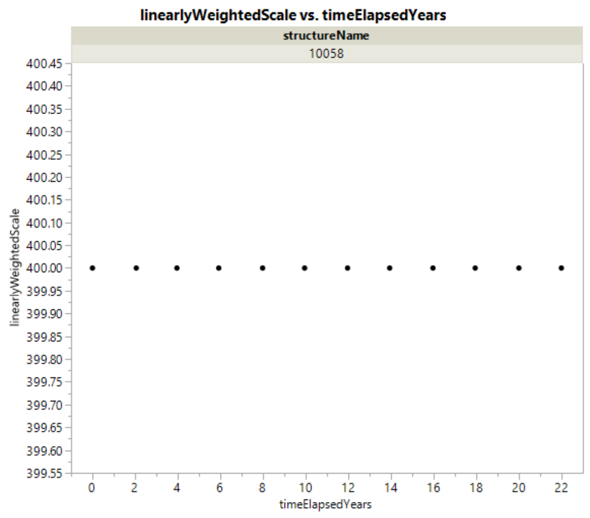
For our next graph, we chose a different element and, with our changes, created the following:



Note that the "score" determined earlier is called "linearlyWeightedScale" here. The x-axis represents how much time has elapsed since the first inspection in years.

This graph shows a very minimal, ever-so-present decrease in LWS (linearlyWeightedScale) as a function of timeElapsed. This was confusing, as the problem made it seem like the correlation would be easy to spot and consistent. We checked to see correlations in individual buildings, and we (generally) found two kinds of graphs. See here:





The left graph confirmed that our approach with LWS had merit and indicated that the problem likely lay in the "structureName" variable, because it was acting as a pseudonym for other variables (it could be any number of things, from weather to renovations to funding) that weren't included, and were making LWS stay constant artificially. Based on this, we came up with the following criteria for a data analysis methodology:

1) We must analyze within a building, and then compare to other buildings (because trends make sense when confined to single buildings)
2) The output must have a heavy emphasis on the time domain, because ultimately IDOT would like to know how their structures are affected over time. Therefore, a solution that can say, in the first year this structure decreases its score on average by 5%, then 8% the second year, then 15% the third year, etc. would be fantastic
3) Account for the fact that some buildings were inspected every few months, and others every two years

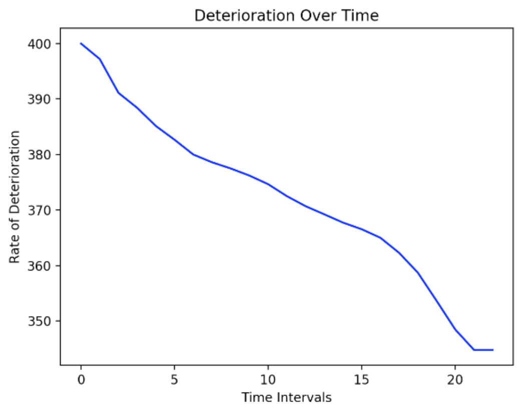
We came up with a couple ideas, and based on the criteria we determined that what we call the "time-window" approach was the best way to go. The following details what that approach does:

1) First pick an interval of time to present the data. We figured a model showing yearly decays would be useful, we made the interval a year.
2) Then pick a window of time that will be used to evaluate the chosen interval. You might think it should match interval, but because we saw several instances where inspections were done every two years, this approach would skip over that data entirely. Also, we thought that building changes during year 1 and year 2 would be similar, so they could be combined into the year 1 average. Year 2 would then include year 2 and year 3, and so on. *This does mean some data is used more than once, so that is a flaw in our compromising approach.*
3) The algorithm will then take all the LWS data for a building in the given time window, and it'll run a linear regression algorithm on the scores. The slope will be averaged with the slopes of all other buildings in that time interval to create an average slope for each time interval. Afterwards, a graph will be constructed from the slopes in each time interval.

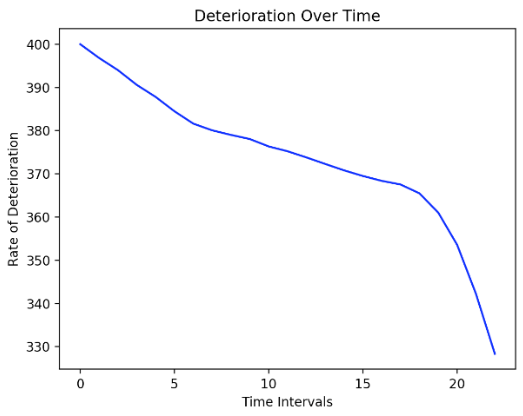
Note: since we had to do a lot of testing and experimenting to get this far, our code runs with an input JSON file that is a proper slice of the data (proper slice being a single element and single environment). These slices can easily be generated in JMP and Excel and fed into the model, and the model itself could be adjusted to parse through the full JSON version of the given data.

You can look at our code for this algorithm at https://github.com/jmather625/Pyghack2018. The python file is "Parser.py," and the various JSON files were created in JMP and analyzed by our code. For all json files listed, the structure is 8026. You can easily create more data slices of the right nature (one structure, one environment key) and analyze it with our code. Below are the graphs and results for the data slices we looked at:

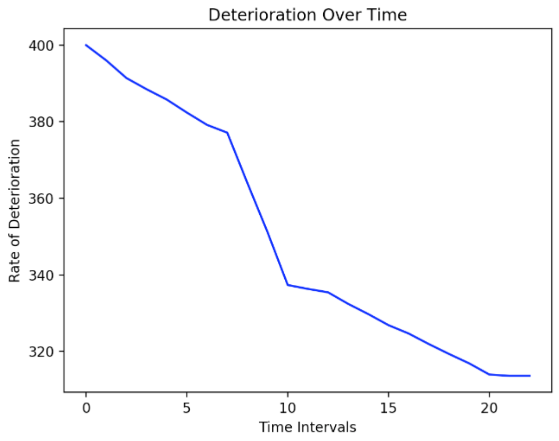
Environment 1:



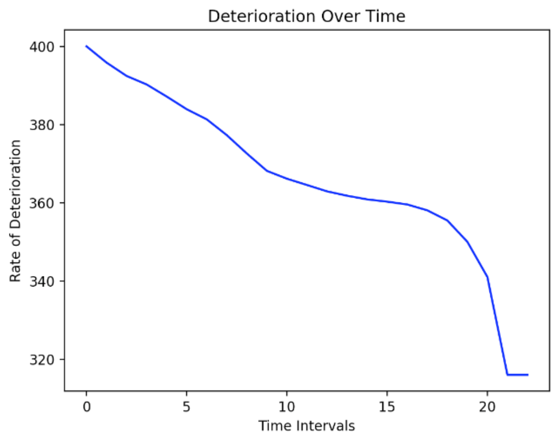
Environment 2:



Environment 3:



Environment 4:



Note that “Rate of Deterioration” is a mislabel; it should read as linearlyWeightedScale on the y-axis. These graphs represent the decay of one structure (element key 8026) in different environments. One can easily imagine doing the same for other element keys sliced by environment. The shapes of these graphs are individually quite interesting (Env. 1 is linear, Env 2. and Env. 4 resemble each other, and Env. 3 decays in an odd manner). After compiling decay graphs for other structures, one could get a better sense for what these are supposed to look like and how accurate our algorithm was. As a final note, to do further analysis you must remember that our code requires JSON files that have proper dictionary keys. We used JMP to make these files, but a better, more versatile way to parse the same slices would be to parse it straight through the code. We are confident your team can come up with a better, more efficient way to do that, but hopefully our code can provide a good starting point.

Contact of Dr. John Sobanjo, professor who helped FDOT: sobanjo@eng.famu.fsu.edu

Contact of team members:

Sudesh Sahu (sksahu2@illinois.edu)

Himanshu Minocha (minocha2@illinois.edu)

Jatin Mathur (jatinm2@illinois.edu)