An Interactive Tool to Visualize Results from Uncertainty Quantification

Matthew Isaac *

Abstract

Uncertainty quantification is a class of methods used to simulate the ways that variations to inputs of a system can potentially change the end state of that system. This paper outlines the implementation of an interactive tool to visualize the results of an uncertainty quantification analysis. The interactive tool is a deployable web application built with the `R Shiny` framework. Users can create, adjust, and save custom visualizations to assist in interpreting and presenting results.

Key Words: Uncertainty Quantification; Statistical Visualization; Interactive Visualization; `shiny` R Package

1. Introduction

Uncertainty quantification is a methodological framework used with some frequency in engineering analysis (Ewing et al., 2018). It is used to understand how variability within the parameters (i.e. inputs) to some system impact the end state of that system. Engineering analysts use uncertainty quantification to assess and find the balance between design sufficiency and design efficiency. This is particularly important in fields where large-scale prototypes, testing, and data collection are extremely expensive. Engineers in these types of applications are increasingly relying on computational simulations to assess system designs.

In the following paper, I outline the implementation of an interactive tool to visualize uncertainty quantification results. The outline of this paper will proceed as follows: In Section 2, a brief overview of uncertainty quantification is given. In Section 3, the methods by which the visualization tool was developed are described. In Section 4, the resulting web application is described and screenshots of the web application are shown. In Section 5, future enhancements to the web application are discussed. Section 6 contains the code written to generate the `shiny` app.

2. Uncertainty Quantification Overview

The following description of the uncertainty quantification algorithm is adapted from Ewing et al. (2018). First, a few terms and definitions will be described.

system response quantity (SRQ): A parameter of particular interest directly related to the engineering system in question. The SRQ is the output (i.e. prediction) from the engineering model.

engineering model: A mathematical model that defines the relationship between the parameters (model inputs) and SRQ (model output).

aleatory uncertainty: Uncertainty resulting from randomness inherent to a given parameter. Gaining more knowledge about the parameter will not reduce the uncertainty of the parameter.

*Department of Mathematics and Statistics, Utah State University, Logan, UT 84322–3900, USA. E-mail: `matt.isaac@aggiemail.usu.edu`

epistemic uncertainty: Uncertainty resulting from a lack of knowledge about a given parameter. Gaining more knowledge about the parameter could reduce the uncertainty of the parameter.

The first step of uncertainty quantification is to identify sources of uncertainty (model parameters) and classify them as either aleatory or epistemic. This classification process has been somewhate debated in literature (Kiureghian and Ditlevsen, 2009), and will not be discussed as it is outside the scope of this project. Once these uncertainties related to the model parameters have been identified and classified as aleatory or epistemic, the uncertainty for each parameter must somehow be described. Traditionally, epistemic uncertainties have been described by an interval over which any value in the interval is equally likely, while aleatory uncertainties have been assigned probability distributions. Some more recent publications (Ewing et al., 2018) propose that all uncertainties, aleatory or epistemic, be described using probability distributions. This debate will not be discussed here as it is, again, outside the scope of this project. Once these distributions and/or intervals have all been assigned, they are carried through the model using Monte Carlo techniques.

Let m denote the number of iterations of an outer for loop, and let n denote the number of iterations in an inner for loop. In the outer for loop, values for the epistemic uncertainty parameters are selected randomly from the intervals/distributions assigned. Upon entering the inner loop, values for the aleatory uncertainty parameters are randomly chosen from the distributions assigned. The values chosen for the parameters in both the outer loop and the inner loop are then used as inputs in the engineering model to calculate a value for the SRQ. This value is stored and the inner loop continues running for the rest of the $n - 1$ iterations. All of the n SRQ values calculated from the n iterations of the inner loop are used to calculate an empirical cumulative distribution function (CDF) of SRQ values and the CDF is stored. The outer loop then begins its second iteration, and new values for the empirical uncertainty parameters are chosen. The inner loop then runs another n iterations, producing another CDF. This process continues until the outer loop has run all of its m iterations.

At this point, there will be m empirical CDFs that have been calculated, representing various possible realized values of the SRQ. These CDFs can then be plotted and interpreted. Ewing et al. (2018) suggests constructing a "P-box". This P-box is found by calculating a lower percentile (e.g. the 5th percentile) and an upper percentile (e.g. the 95th percentile) of the CDF ensemble. This P-box can then be interpreted in several ways, including (1) selecting a SRQ value and extracting a probability interval, and (2) selecting a probability value and extracting an SRQ interval.

3. Methods

In order to assist analysts in visualizing and interpreting results from an uncertainty quantification analysis as described in Section 2, an interactive visualization tool was developed. Since the actual uncertainty quantification analysis can be carried out with greater speed and computational power elsewhere, this implementation does not include the Monte Carlo portion of the implementation described in Section 2. The web application was built using the `shiny R` package and the code for this application is contained in two files. The first file, `ui.R` (short for 'user interface') contains the code that controls the layout of the various panels and panes in the application, as well as the placement of the user interface elements (i.e. text inputs, sliders, etc.). The `server.R` file contains the computational code that performs calculations, data manipulation, and generates the plots. See Section 6 for the source code.

3.1 R Packages

The following packages and methods were used in the development of the web application:

dplyr: The `dplyr` package (Wickham et al., 2013) is a data-wrangling and manipulation package implemented in `R`. The methods in this package were used to manage and manipulate the CDFs from the `.csv` file into a convenient format for plotting.

shiny: The `shiny` package and framework (Chang et al., 2018) constitues the backbone of this project. `shiny` provides a way to create and deploy web applications through RStudio (RStudio Team, 2016). It also contains the implementations for all user-interface components (toggle buttons, check boxes, numeric inputs, sliders, etc.). The `shinydashboard` package (Chang and Borges Ribeiro, 2018) was also used as an aesthetic wrapper around the `shiny` framework. The `shiny` package was be used to create and will be used to deploy the web application.

ggplot2: The plotting functionality of the `ggplot2` package (Wickham, 2016) was used to generate the actual visualization and to add, remove, or adjust components on the plot.

plotly: The `plotly` package (Sievert, 2018) is not currently used in the implementation, but may be used to add additional interactive capabilities to the plot. `plotly` includes a method called `ggplotly()` which could be used to convert the `ggplot` plot object to a `plotly` plot object. The `plotly` plots contain options to zoom in and out on a plot, show plot values when hovering with mouse, download and save a `.png` version of the plot, and download and save an interactive `html` version of the plot.

Since the existing `R` packages already implemented most of the tools needed to generate the plot and implement user interface elements, the primary task on this project was to seamlessly combine elements from the packages above (particularly the `shiny`, `ggplot2`, and `plotly` packages) to create a user-friendly interactive visualization tool.

4. Results

4.1 Features

The resulting web application currently has the several features implemented. First, the capability was added to allow the user to upload their own data from a `.csv` file to be used in the visualization. If users want to experiment with the visualization but don't have their own data readily on hand, they can also choose to use some provided sample data. Once the data source is specified by the user, several options are provided to allow customization of the visualization. Users can show or hide the P-box, as well as specify the upper and lower percentiles to be used in the P-box calculation. The CDF ensemble can also be toggled on and off so only the P-box is shown. Oftentimes there are hundreds or thousands of CDFs to be plotted, so naturally a lot of overplotting can occur. A slider is included in the control panel to adjust the tranparency of the CDFs. A download button has also been provided so users can download and save a `.png` file of the current rendering of the plot.

A few preliminary screenshots from the web application are included in this paper. Figure 1 shows downloaded plots with various plotting options selected (show/hide P-box and CDFs). Figure 2 shows a full-view screenshot of the application.

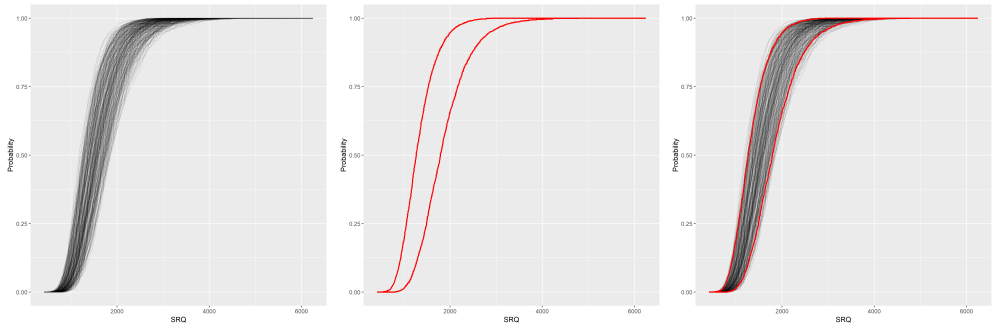

Figure 1: Plots downloaded and saved from the web application.

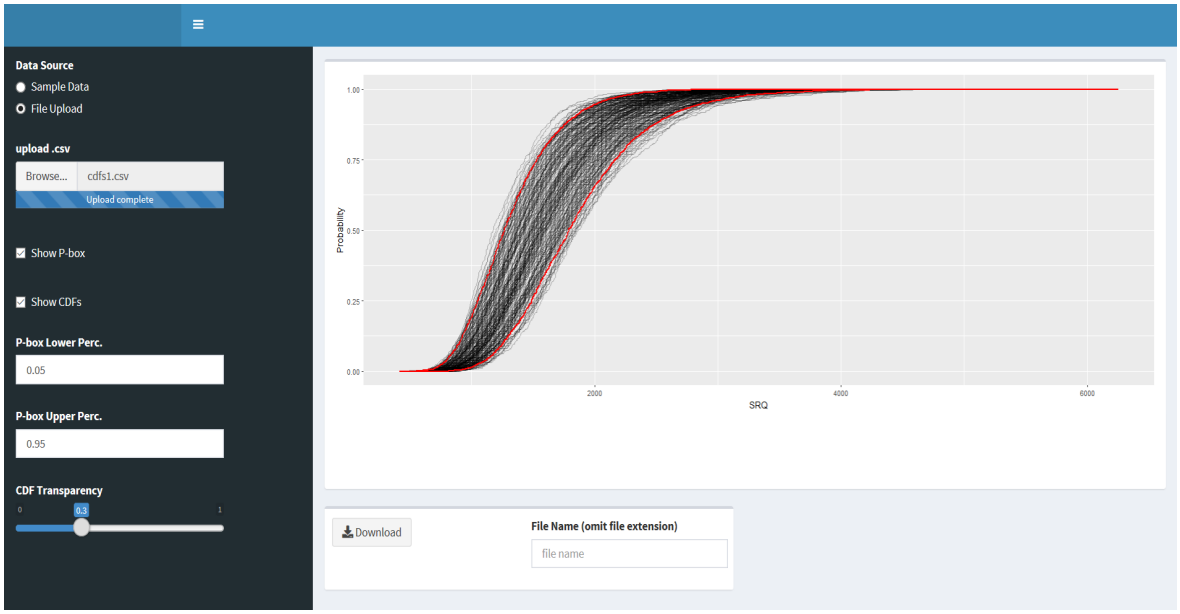

Figure 2: Full screenshot.

4.2 Challenges

One main challenge with this visualization tool that was unanticipated was the time that it would take to render the plot. The nature of uncertainty quantificataion and the nature of this kind of visualization leads to there being hundreds or even thousands of traces on a single plot. In addition, the way the `shiny` package is implemented causes the plot to completely re-render when user inputs change. Thus, whenever the user selects a new plotting option, he or she must wait several seconds (depending on the size of the data set being used) for the plot to re-render. This leads to a less 'interactive' feel for the user. As this application is still under development, possibilites for decreasing the re-rendering time of the plots are being explored.

5. Future Work

In the next two weeks I plan to fine-tune the visualization tool and produce a polished final product. There is also one more major feature to implement that will greatly aid in the interpretabilty of the visualization. This feature is the ability for a user to input an SRQ

value and extract a probability range, as well as the ability to input a probability and extract an SRQ range. Finishing touches will also include adjusting the spacing between controls to improve aesthetics and adding a user manual/help page.

6. Appendix

`R` code for `shiny` web application.

6.1 `ui.R`

```
library(shiny)
library(shinydashboard)
library(plotly)

dashboardPage(
  dashboardHeader(),
  dashboardSidebar(
    width = 400,
    radioButtons(inputId = 'radioDataSource',
                 label = 'Data Source',
                 choices = c("Sample Data",
                             "File Upload"),
                 selected = "Sample Data"),
    fileInput(inputId = "fileIn", label = "upload .csv"),
    checkboxInput(inputId = "checkPbox",
                  label = "Show P-box",
                  value = TRUE),
    checkboxInput(inputId = "checkCDFs",
                  label = "Show CDFs",
                  value = TRUE),
    numericInput(inputId = "pboxLower", label = "P-box Lower Perc.",
                 value = 0.05),
    numericInput(inputId = "pboxUpper", label = "P-box Upper Perc.",
                 value = 0.95),
    sliderInput(inputId = "sliderAlpha", label = "CDF Transparency",
                value = 0.3,
                min = 0,
                max = 1,
                step = 0.1,
                ticks = FALSE)
  ),
  dashboardBody(
    fluidRow(
      box(
        width = 12,
        height = 500,
        # main plot
        plotOutput("uqPlot")
      )
```

```
    ),
    fluidRow(
      box(
        splitLayout(
          downloadButton(outputId = 'savePng'),
          textInput(inputId = "pngName", value = "", placeholder = "file name
        )
      )
    )
  )
)
```

6.2 server.R

```
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(ggplot2)
library(plotly)
library(dplyr)

################################################
# Functions ####################################
################################################
plotAllLayers <- function(df){
  p<-ggplot(data = df, aes(df[,1]))
  for(i in names(df)[-1]){
    p<-p+geom_line(aes_string(y=i), alpha = 0.2)
  }
  return(p)
}

calc_pbox <- function(data, p_upper, p_lower){
  # for row in np.transpose(cdf_arr): # loop over transposed matrix
  #   qlower = np.quantile(row, q = pbox_lower) # calculate lower quantile
  #   qupper = np.quantile(row, q = pbox_upper) # calculate upper quantile
  #   ql[counter] = qlower # store
  #   qu[counter] = qupper # store
  #   counter = counter + 1
  pbox <- apply(values$data, 1, FUN = quantile, probs = c(p_lower, p_upper))
  # pbox <- data_frame(pbox)
```

```
  return(t(pbox))
}

################################################
# Reactive Values###############################
################################################

values <- reactiveValues()

################################################
# Read in Sample Data ##########################
################################################
cdf_arr <-  read.csv('data/cdfs.csv', header = FALSE)
cdf_arr <-  t(cdf_arr)
cdf_df <-  data.frame(cdf_arr)

################################################
# Server Logic #################################
################################################
shinyServer(function(input, output) {

  output$uqPlot <- renderPlot({
    if(input$radioDataSource == "Sample Data"){
      values$data <- cdf_df
    } else {
      req(input$fileIn)
      df <- read.csv(input$fileIn$datapath,
                     header = FALSE)
      df <- data.frame(df)
      values$data <- df
    }

    req(values$data)
    plt <- ggplot(data = values$data, aes(values$data[,1]))
    if(input$checkCDFs == TRUE){
      for(i in names(values$data)[-1]){
        plt <- plt + geom_line(aes_string(y=i), alpha = input$sliderAlpha)
      }
    }
    plt <- plt +
      xlab("SRQ") +
      ylab("Probability")

    if(input$checkPbox == TRUE){
      pbox <- calc_pbox(values$data, (1 - input$pboxLower), (1 - input$pboxUp
      plt <- plt +
        geom_line(aes(y = pbox[,1]), col = "red", lwd = 1) +
        geom_line(aes(y = pbox[,2]), col = "red", lwd = 1)
    }
```

```
    values$plot <- plt

    plt
  })

  # help from https://stackoverflow.com/questions/14810409/save-plots-made-in
  output$savePng <- downloadHandler(
    filename = function() { paste(input$pngName, '.png', sep='') },
    content = function(file) {
      ggsave(file, plot = values$plot, device = "png")
    }
  )

})
```

References

Chang, W., Borges Ribeiro, B., 2018. shinydashboard: Create Dashboards with 'Shiny'. R package version 0.7.1 (https://CRAN.R-project.org/package=shinydashboard).

Chang, W., Cheng, J., Allaire, J., Xie, Y., McPherson, J., 2018. shiny: Web Application Framework for R. R package version 1.2.0 (https://CRAN.R-project.org/package=shiny).

Ewing, M., Liechty, B. C., Black, D., 2018. A General Methodology for Uncertainty Quantification in Engineering Analyses Using a Credible Probability Box. Journal of Verification, Validation, and Uncertainty Quantification 3 (2), https://doi.org/10.1115/1.4041490.

Kiureghian, A. D., Ditlevsen, O., 2009. Aleatory or Epistemic? Does it Matter? Structural Safety 31 (2), http://www.sciencedirect.com/science/article/pii/S0167473008000556.

RStudio Team, 2016. RStudio: Integrated Development Environment for R. RStudio, Inc., Boston, MA, (http://www.rstudio.com/).

Sievert, C., 2018. plotly for R. (https://plotly-book.cpsievert.me).

Wickham, H., 2016. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, (http://ggplot2.org).

Wickham, H., Francois, R., Henry, L., Muller, K., 2013. dplyr: A Grammer of Data Manipulation. R package version 0.7.6. (https://CRAN.R-project.org/package=dplyr).