Team #7 – Final Project Documentation

Aidan Bjelland, Inrae Cho, Caroline Dolbear

Project Objective and Overview:

Create a python code that reads an Excel spreadsheet as an input file, computes necessary calculations to create a bridge model, generates that model in Abaqus, and then analyzes that model.

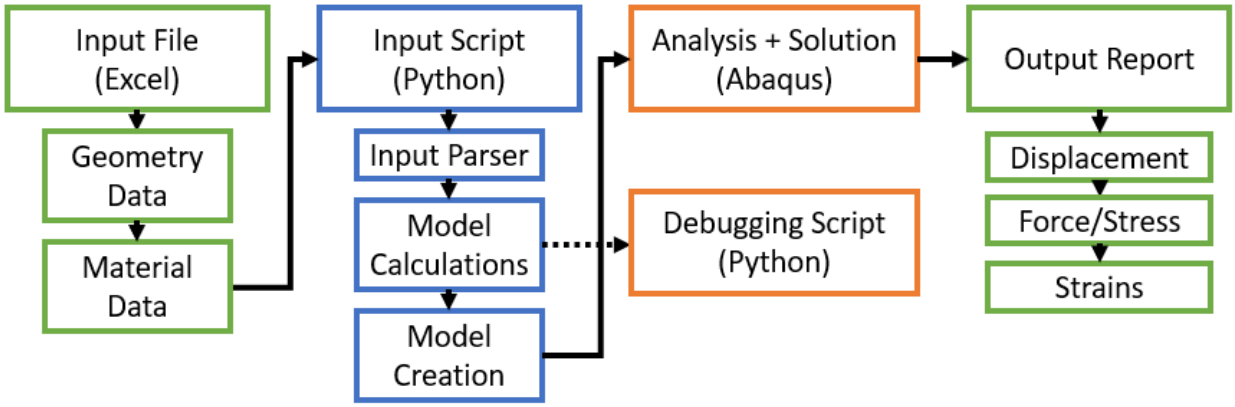


Figure 1. Project Overview.

Required Packages Packages:

- Csv: Needed for input parser
- Abaqus: Internal scripting interface - converts Python script to Abaqus model
- Matplotlib: Plotting bracing
- Numpy: Data manipulation
- Math: Data manipulation

Note - *A valid installation of Abaqus CAE 2021 (or newer) is required to utilize the full Python script (see end of document for installation instructions).*

Task Contributions:

Table 1. Task Delegation.

Tasks:	Aidan	Inrae	Caroline
Input File	40%	30%	30%
Input Parser	15%	85%	0%
Model Calculations	0%	50%	50%
Abaqus Macro Script	100%	0%	0%
Input Debug Script	0%	0%	100%

Bridge Nomenclature

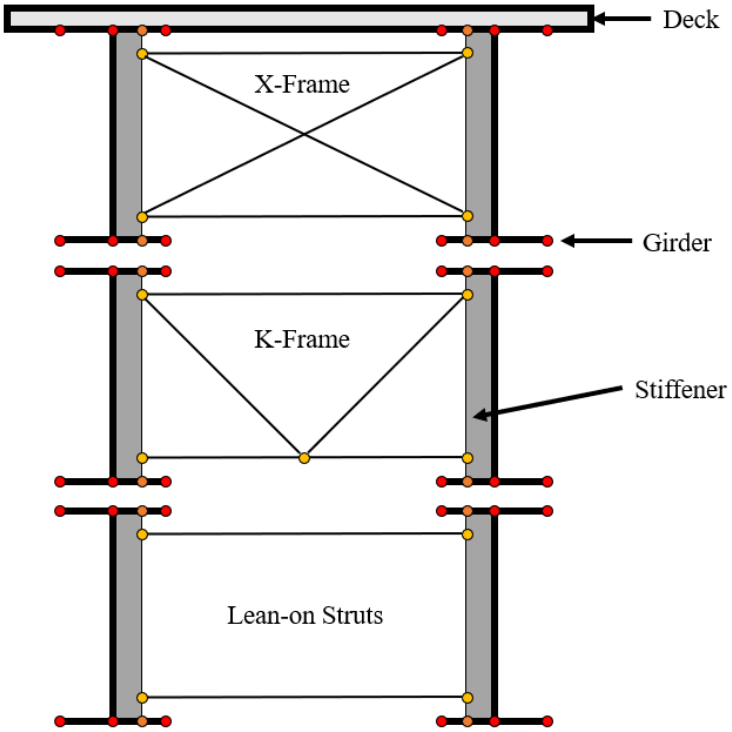


Figure 2. Different bracing type options.

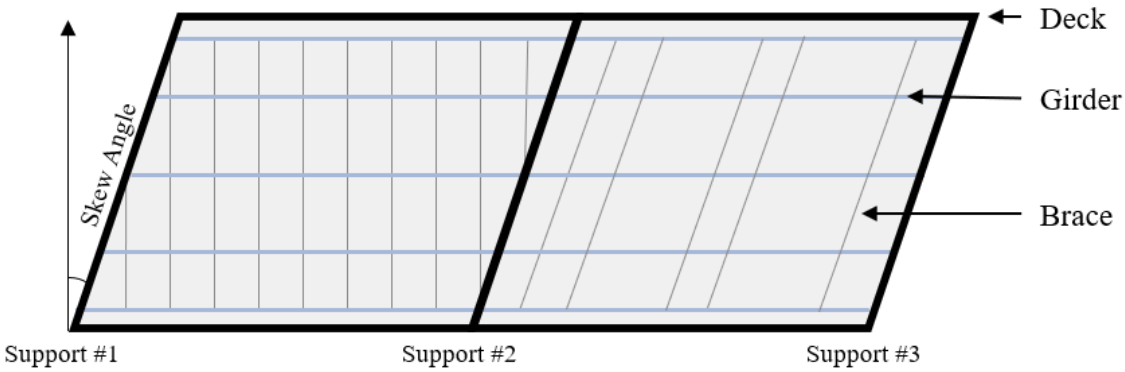


Figure 3. Visual representation of bridge terminology.
Note: Bridge will be completely uniform/nonuniform and normal/parallel, cannot splice these options together in different spans. The graphic is just a visual representation.

Input File Format

Table 2. Input File Formats.

Input:	Group:	Note:
ID	Label	(int)
Amount of Girders	Girder	(int)
Span Length		[List], (float)
Cross Section Order		[List, List, ...], (float)
Cross Section Property		[List, List, …], (float)
Splice Location		[List, List, …], (float)
Girder Spacing		[List], (float)
Girder Type		String
Skew	Bridge Modifiers	(float)
Steel Properties		[List], (float)
Deck		[List], (float)
Bracing Configuration	Bracing	Uniform or Nonuniform
Number of Braces		(int) or [List] or [List, List, …]
Spacing of Braces		(int) or [List] or [List, List, …]
Brace Orientation		Parallel or Normal to Support (Based upon Skew)
Brace Type		String
Stiffener Offset		[List], (float)
Brace Properties		[List], (float)
Stiffener Properties		[List], (float)
Support Buffer	Supports	(float)
Support Type		[List], (int)

Input Preprocessor Documentation

Input Data Parser Function:

- **system_iden.retrieve (filename, i = 3):** Retrieve function takes filename and row number that user wants to analyze as input. First, input data parser pulls raw data from the input csv file using the CSV python package. Initial format of the data will either be a list of strings or a string. Using defined format correcting functions, input data parser will convert string into the desired formats. Then, it will store the converted data in dictionary format.
    - Package used: csv
- **FormatCorrector (ListofLists):** Take string of list of lists to list of lists containing proper float entries
- **FormatCorrector2 (List):** Take list of strings and converts into list of proper float

Geometry Calculation Function(s):

- **GirderSketch(xprops, xtype):** This function takes in a list of cross-section properties [flange thickness, flange width, web thickness, web height] and cross-section type of the girder cross-section. The function returns the node positions for the cross-section and node connectivity between the nodes.
- **system_iden.splice_local (System_info):** This function takes system information obtained from 'retrieve' function as an input. It calculates local coordinates of splices within a girder.
- **system_iden.normal_bracing (System_info):** This function can only be used for "Normal" bracing conditions. It takes system information obtained from the 'retrieve' function as an input. It calculates and assigns global x and y coordinates of each braces for all girders of the interested bridge. This function returns x and y coordinates of braces as a tuple of two lists of lists format.
    - Package used: Math
- **system_iden.parallel_bracing (System_info):** This function performs the bracing coordinate calculations for the "Parallel" bracing orientation. It takes the system information dictionary and calculates the uniform or nonuniform bracing positions for the base girder, then defines the remaining bracing positions on the other girders using numpy and math to offset using the skew and girder spacing provided. This function returns the x and y coordinates of the braces in separate lists, with each x or y list containing lists of coordinates per girder.
    - Packages used: Numpy, math
- **system_iden.splice_dist_calc (System_info):** This function takes local splice coordinates calculated from 'splice_local' function and a list of two splice points that the user wants to calculate distance between. It calculates and returns the distance between the two points in float format.

Input Debug Documentation

Debugging / Error Check:

- **DEBUG_Plotting(bracing_xcoord, bracing_ycoord, System_info):** This function is located in a separate DEBUG script so that Abaqus does not compile matplotlib when running the main script. The function takes the bracing coordinates and system information for the bridge, and plots the bracing coordinates, braces, girders, and endpoints of the bridge to allow the user to visually inspect the bridge input prior to running Abaqus. Two separate plotting algorithms are used for “Normal” vs. “Parallel” bracing orientations.
    - Packages used: Numpy, math, matplotlib.pyplot

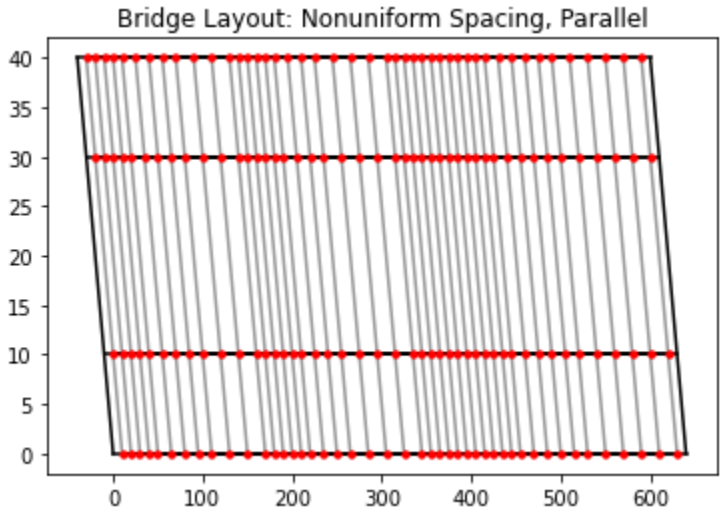


Figure 4. Example DEBUG_Plotting output.

- **DEBUG_DataErrors(System_info):** This function is also in the DEBUG script and is intended to be used prior to Abaqus to catch any irregularities in the input data. This function contains many if statements and raises a custom exception based on the error found. There are four classes of errors: DataTypeError, UnacceptedEntry, ListLengthError, and DataError. DataTypeError screens for incorrect data types in the System_info dictionary. ListLengthError screens for expected list lengths for entries with defined inputs. UnacceptedEntry errors utilizes lists of acceptable strings for string entries, and if the entry is not in the list it informs the user of the acceptable strings. DataError screens for incompatible data within the dictionary, like if the splice distance is greater than the girder length.
    - Package used: Numpy

Abaqus Macro Documentation

Abaqus Functions:

- **Main():** Serves to run the functions in a set sequence, handle minor Abaqus calls (analysis execution) and provide additional inputs as necessary.
    - Packages used: abaqus*

Sketch / Part Functions

- **IBeamSketch(s, PList, CList):** Creates the planar sketch for the I-Beam cross-section based on a set of points (PList) and connections of those points (CList).
    - Packages used: abaqus*

- **DeckSketch(System_info, PartName):** Creates the planar sketch for the deck. Converts this sketch into a part.
    - Packages used: abaqus*

- **StiffenerBraceSketch(GSpacing, System_info, ii):** Creates the planar sketch of the stiffeners and struts. Converts these sketches into parts for normal bridges.
    - Packages used: abaqus*

- **StiffenerBraceSketch2(GSpacing, System_info, ii):** Creates the planar sketch of the stiffeners and 3D sketch of the wires. Converts these sketches into parts for parallel bridges.
    - Packages used: abaqus*

- **DatumPartition(PLocation, Plane, PartName, ii):** Used to divide part geometry into distinct sections.
    - Packages used: abaqus*

- ***DatumPartitionS(PLocation, ii, PartName):*** Used to divide part stiffeners into distinct sections *(currently unused, may have future use)*.
    - Packages used: abaqus*

- **PartBeamExtrude(i, GLength):** Extrudes beam cross-section into a beam part of a specified depth.
    - Packages used: abaqus*

Section / Property Assignment Functions

- **BridgeMaterial(System_info):** Defines materials utilized in bridge (steel, concrete).
    - Packages used: abaqus*

- **CreateBridgeSections(System_info):** Creates sections and their properties for each bridge element (shell and truss property).
    - Packages used: abaqus*

- **BeamAssignment(System_info, splice_coord, PartName):** Assign created beam sections to corresponding components in the bridge model.
    - Packages used: abaqus*

- **BracingAssignment(SPartName, BPartName, SSectionName, BSectionName):** Assign created stiffener and strut sections to corresponding components in the bridge model.
    - Packages used: abaqus*

- **DeckAssignment(System_info, PartName, SectionName):** Assigns created deck section to corresponding component in the bridge model.
    - Packages used: abaqus*

Assembly Functions

- **BraceAssembly(ii):** Creates two stiffener instances and a strut instance and assembles them into a single part. This is for normal bridges.
    - Packages used: abaqus*

- **BraceAssembly2(ii, System_info):** Assembles two stiffener parts and a strut part into a single part. This is for parallel bridges.
    - Packages used: abaqus*

- **SuperstructureAssembly(GLocation, BCoord, System_info):** Creates ‘n’ girder instances and ‘m’ bracing system instances. These are then assembled into a single part.
    - Packages used: abaqus*

- **BridgeAssembly(System_info):** Creates an instance of the superstructure part and the deck park.
    - Packages used: abaqus*

Analysis Functions

- **TieInteraction(System_info, GLocation):** Creates a tie interaction between a region of the deck and corresponding top flange of the girder.
    - Packages used: abaqus*

- **Supports(System_info, GLocation):** Creates roller or pin supports along edges of each girder at the support locations.
    - Packages used: abaqus*

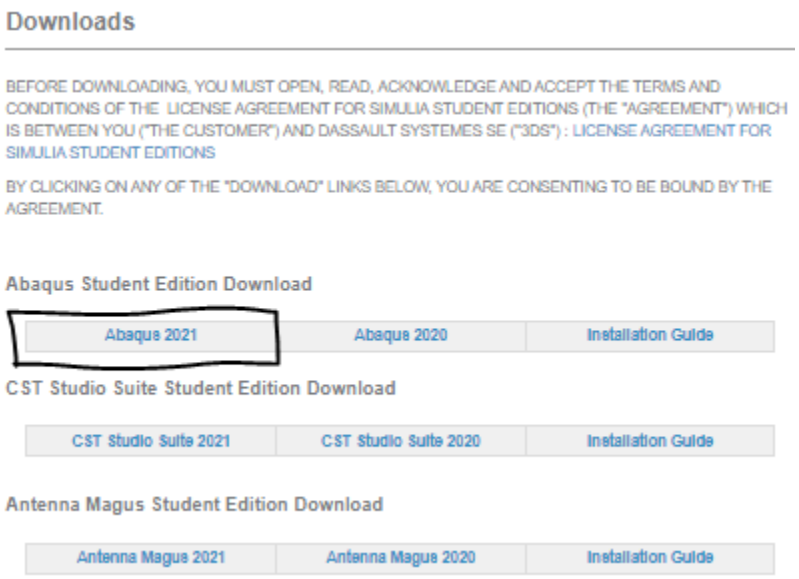
- **Mesh(System_info):** Assigns shell element and truss element mesh density and type.
    - Packages used: abaqus*

* Abaqus contains many internal packages. It is difficult to distinguish between these for the most part.

Abaqus Installation

Follow the link below and create an account. After creating an account, log in and download the Abaqus Student Edition (Version 2021).

Link: https://edu.3ds.com/en/software/abaqus-student-edition



NOTE: *The student version is limited to 1000 nodes. Larger models will be built (95% of the code will run successfully), but the analysis will not be run.*

Code Instructions

We suggest using the InputDebug.py function first, the Demo version of this can be found in a folder with a Demo csv file with some errors. To use InputDebug.py, just enter the csv file name as the InputFileName and select the row your bridge is stored in InputRow. In the InputDebug.py file you can select between 0, 1, and 2 for debugging mode (Note: If 2 is chosen and an error is found, the plotting function will not be executed).

Once the bridge you want to run has been verified using InputDebug.py, you can open AbaqusMacro.py and enter the desired Abaqus working directory, InputFile, and InputRow. From there, open Abaqus and go to File -> Run Script and select AbaqusMacro.py. To view the full model, select the assembly under the assembly tab, you do this by left clicking twice on the tab. To see the analysis of a bridge with less than 1000 nodes, you go to the Results tab in the top left of Abaqus, and then you can click on whichever display group you want. To view these results as an excel file, find the excel file in the working directory.

Abaqus Interface