CHAPTER 8

APPENDIX A: COMPLETE CODE LISTING

CHAPTER 7

REFERENCES

[1] *ASME B30.20 Below-the-Hook Lifting Devices.* Two Park Avenue, New York, NY 10016-5990: American Society of Mechanical Engineers, 2015.

[2] *ASME BTH-1 Design of Below-the-Hook Lifting Devices.* Two Park Avenue, New York, NY 10016-5990: American Society of Mechanical Engineers, 2015.

[3] Steven C. Chapra. *Applied Numerical Methods with MATLAB for Engineers and Scientists.* second. McGraw-Hill, 2008.

[4] Dictionary.com. *Optimization — Define Optimization at Dictionary.com.* 2019. URL: `https://www.dictionary.com/browse/optimization`.

[5] Andries P. Engelbrecht. *Computational Intelligence: An Introduction.* second. John Wiley and Sons, Ltd, 2007.

[6] Justin M. Hughes. *Latin Hypercube Sampling (LHS).* 2017. URL: `https://icme.hpc.msstate.edu/mediawiki/index.php?title=Latin_Hypercube_Sampling_(LHS)`.

[7] M. D. McKay, R. J. Beckman, and W. J. Conover. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code". In: *Technometrics* 21.2 (1979), pp. 239–245. ISSN: 00401706. URL: `http://www.jstor.org/stable/1268522`.

CHAPTER 6

CONCLUDING REMARKS

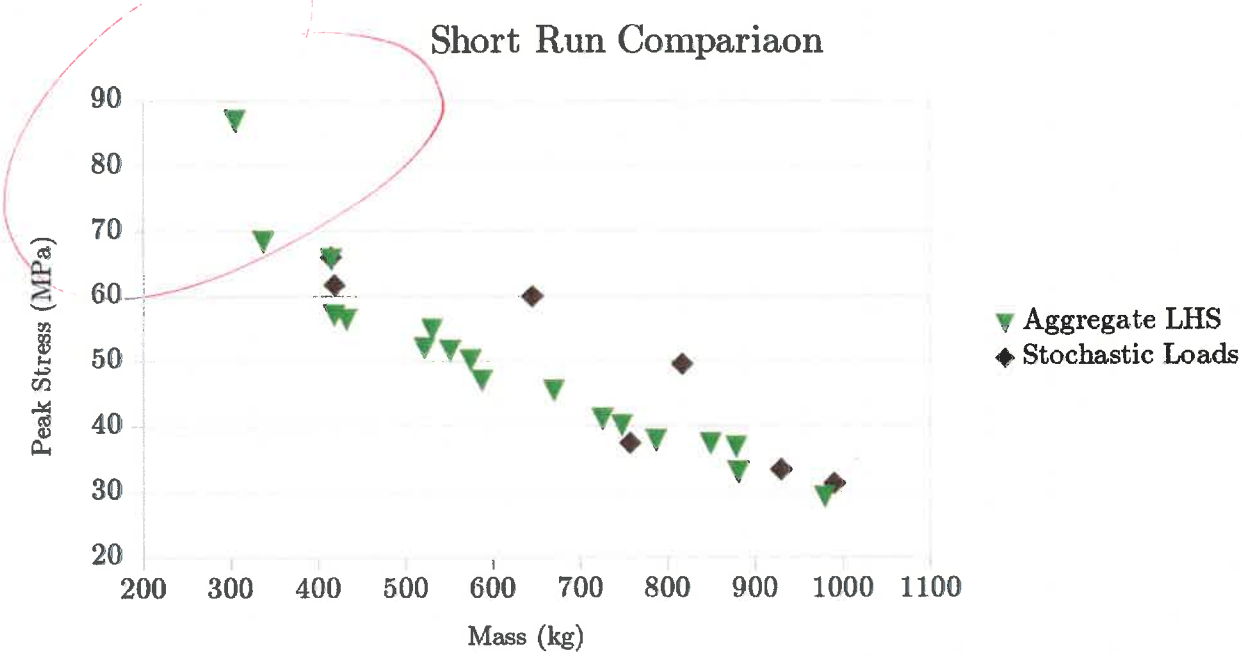


Figure 11: Comparison of the Two Short Run Pareto Plots

Is this a cut of Fig. 10?

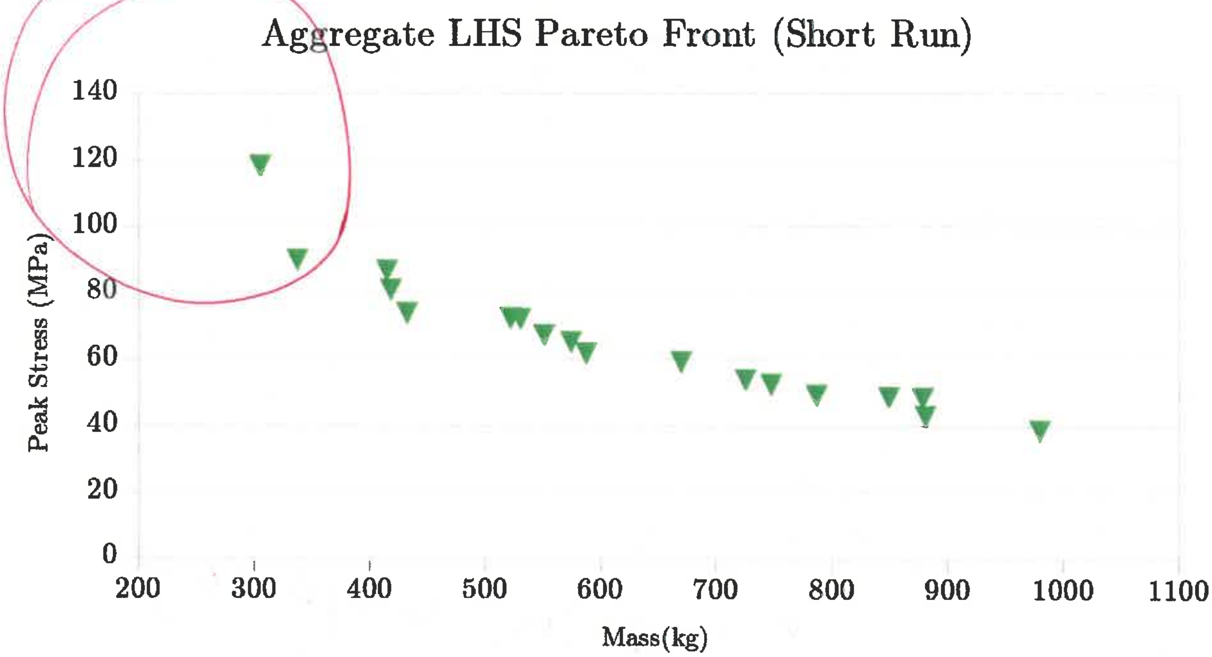


Figure 10: Graph of the Pareto Front generated through Aggregated LHS (Short Run)

Solution Statistics

This solution also was tracked for several computing performance indicators to compare the performance of the algorithm with the other solutions. These statistics are listed in Table 8.

Total Generations Computed	50
Average Time Per Generation (sec)	8.22
Total Wall Clock Time (sec)	411

Table 8: Solution Statistics for Aggregated LHS (Short Run)

5.2.3 Comparison

In order to compare the two pareto fronts in the results above, each design was exposed to an identical load consisting of a 147 kN force acting vertically along the y axis. The peak stress was obtained and tabulated. Figure 11 shows a plot of this comparison between the two fronts. Note that despite the same solution times, the Stocastic Loads plot has resulted in designs returning higher stress values in some weight ranges.

Compare ~~to~~ Aggregated LHS long & short runs

~~Compare~~ Stochastic method long & short runs

Total Generations Computed	25
Average Time Per Generation (sec)	18.1
Total Wall Clock Time (sec)	453

Table 6: Solution Statistics for Stochastic Loads (Short Run)

5.2.2 Aggregate LHS Method

This solution was calculated using parameters designed to produce approximately the same wall clock solution time as the Stochastic Loads Short Run above. The command line that was used was:

```
python -m pystruct <<data_file>>  -S2 -i20 -g2 --csv
```

Where:

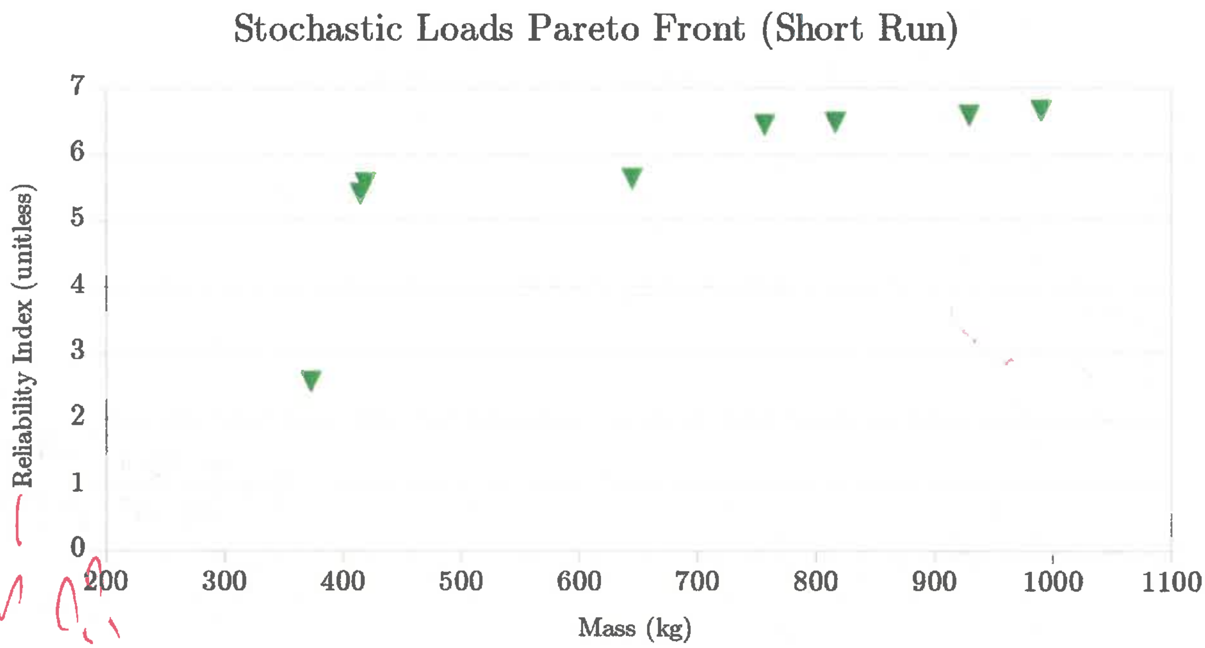
- -S2: Select solution 2: Aggregate LHS method
- -i20: Use a population size of 20.
- -g2: Use a generation count of 2. ? not long enough to get convergence
- --csv: Generate CSV output for final Pareto Front

Resultant Pareto Front

Table 7 shows the design parameters of the members of the Pareto Front generated by this solution run. The Pareto Front is also given graphically in Figure 10.

Parent Load Case	Top Flange Width	Bottom Flange Width	Web Thickness	Doubler Thickness at Hoist Pin	Doubler Thickness at Load Pin	Peak Stress	Mass
	mm	mm	mm	mm	mm	MPa	kg
5	129.488	178.7	50.012	64.189	66.968	65.185	574.583
6	49.705	155.799	78.425	32.802	132.61	52.474	747.643
6	151.264	247.119	22.268	102.256	62.329	73.893	432.479
6	99.534	146.74	64.76	52.983	244.566	53.820	725.557
7	142.734	225.029	57.877	144.821	224.451	49.166	786.819
8	74.907	224.589	54.305	48.5	59.581	61.755	587.581
10	215.954	234.061	88.335	55.669	231.921	38.577	979.637
11	140.513	241.439	18.337	53.885	203.955	80.989	418.511
12	178.503	194.828	43.687	147.502	187.016	59.195	669.831
12	206.098	236.439	75.92	91.15	157.782	42.924	880.824
12	160.811	167.923	69.169	134.097	230.611	48.597	849.269
13	81.25	202.425	42.572	109.538	81.68	67.260	551.180
17	23.721	163.915	49.492	31.35	127.582	72.366	522.037
18	140.976	128.319	39.838	34.495	234.181	72.149	530.609
18	102.296	214.843	13.829	47.973	164.341	89.932	337.531
18	49.437	171.257	79.644	147.498	193.402	48.320	878.472
19	54.821	251.784	24.932	117.632	33.267	86.715	415.135
22	52.03	186.161	10.735	57.554	199.908	118.283	305.286

Table 7: Members of the Pareto Front generated through Aggregated LHS (Short Run)



Green ?

Figure 9: Graph of the Pareto Front generated through Stochastic Loads (Short Run)

Resultant Pareto Front

Table 5 shows the design parameters of the members of the Pareto Front generated by this solution run. The Pareto Front is also given graphically in Figure 9.

Top Flange Width	Bottom Flange Width	Web Thickness	Doubler Thickness at Hoist Pin	Doubler Thickness at Load Pin	Reliability Index	Mass
mm	mm	mm	mm	mm	ul	kg
170.039	245.14	71.477	192.605	189.368	6.578	929.701
61.042	223.528	106.182	54.387	103.309	6.651	990.429
248.513	224.849	77.891	30.268	54.893	6.473	816.670
197.478	184.497	17.633	92.008	136.953	5.407	414.890
103.886	16.745	16.36	158.441	185.596	2.562	372.892
230.115	176.491	20.515	21.348	203.78	5.562	419.033
43.425	101.887	53.24	200.759	78.852	5.628	645.181
143.639	222.302	73.671	32.004	87.213	6.437	757.066

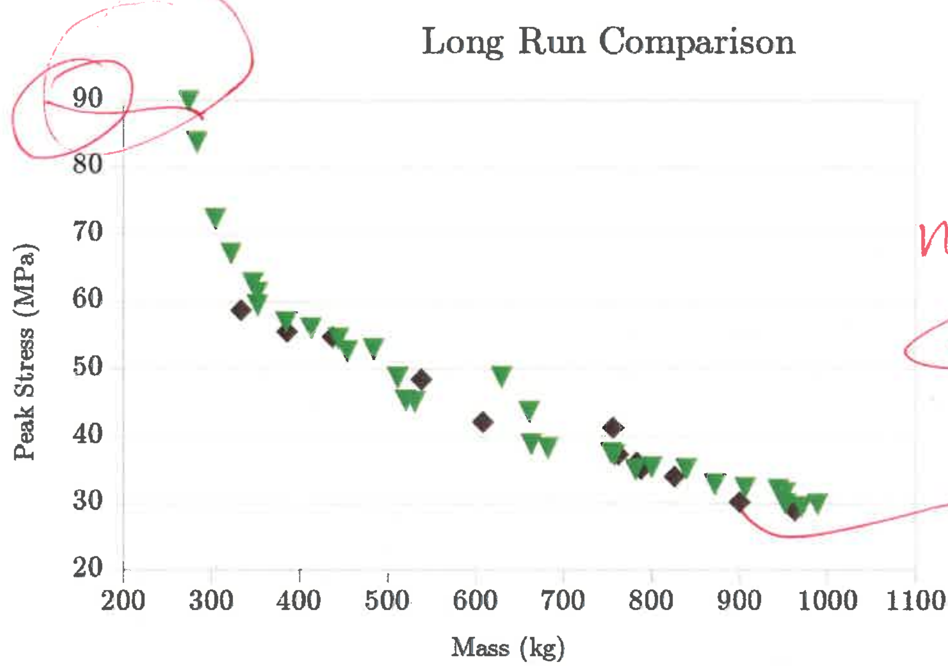
Table 5: Members of the Pareto Front generated through Stochastic Loads (Short Run)

Solution Statistics

This solution also was tracked for several computing performance indicators to compare the performance of the algorithm with the other solutions. These statistics are listed in Table 6.

5.1.3 Comparison

In order to compare the two pareto fronts in the results above, each design was exposed to an identical load consisting of a 147 kN force acting vertically along the y axis. The peak stress was obtained and tabulated. Figure 8 shows a plot of this compairson between the two fronts. Note that the solver for the aggregate LHS appears to cover a wider portion of the solution space, but that both graphs follow roughly the same curve.



not the same as that presented in fig. 7

How to get it from reliability

Figure 8: Comparison of the Two Long Run Pareto Plots

5.2 Short Runs

5.2.1 Stochastic Loads Method

This solution was run with a targeted execution time of approximately 7-8 minutes. The solution was calculated using the following command line arguments to the solver:

```
python -m pystruct <<data_file>>  -S3 -i30 -g25 --csv
```

Where:

- -S3: Select solution 3: Stochastic Loads Method
- -i30: Use a population size of 30.
- -g25: Use a generation count of 25. Convergence?
- --csv: Generate CSV output for final Pareto Front

Parent Load Case	Top Flange Width	Bottom Flange Width	Web Thickness	Doubler Thickness at Hoist Pin	Doubler Thickness at Load Pin	Peak Stress	Mass
	mm	mm	mm	mm	mm	MPa	kg
0	196.196	238.495	17.894	32.068	122.72	75.321	385.017
2	212.903	233.733	18.028	27.453	184.538	72.971	413.313
2	130.786	239.166	42.033	24.93	118.138	58.741	530.927
3	111.472	228.83	23.545	29.255	41.03	80.908	352.336
4	78.553	244.856	68.939	67.863	203.348	48.071	781.942
5	9.233	224.536	103.361	41.51	115.829	41.160	944.191
5	61.752	150.76	21.663	28.219	35.563	107.743	283.783
6	238.215	208.036	57.469	65.061	236.733	48.752	757.997
7	218.927	248.639	19.067	30.017	238.979	70.697	454.470
7	231.66	245.967	74.31	84.323	144.349	44.088	871.946
8	115.387	191.202	51.482	95.676	188.118	56.802	660.743
8	97.613	236.443	98.833	50.915	99.909	41.0519	951.539
11	67.197	227.432	85.794	54.263	95.015	46.108	839.268
11	91.587	255.03	98.591	74.331	136.942	39.216	988.806
11	201.384	211.146	62.053	72.464	165.169	49.240	754.091
12	106.143	246.724	96.2	46.847	180.653	40.340	970.445
12	182.514	239.526	91.69	38.44	166.629	40.940	952.231
13	62.208	129.097	22.12	22.923	35.153	117.951	274.318
13	219.516	246.508	47.023	44.833	214.989	50.956	663.225
15	236.696	245.392	35.828	23.168	99.292	61.540	520.578
16	93.266	186.468	16.423	20.836	126.832	96.200	304.634
16	136.649	203.085	39.35	26.632	144.851	64.828	511.432
16	78.254	243.559	60.134	42.531	170.557	50.865	682.102
16	53.262	238.343	69.145	94.239	224.036	47.076	800.102
18	1.72	213.492	91.354	62.696	216.3	41.440	906.493
18	46.269	200.365	24.106	20.156	66.847	85.989	322.573
18	171.216	190.008	30.221	19.063	234.219	67.857	484.007
19	155.298	235.957	52.813	70.096	57.602	58.121	629.524
19	145.627	204.172	28.536	31.343	158.165	72.446	444.387
19	231.695	188.299	8.604	50.656	23.371	108.290	282.637
22	158.571	235.695	17.301	26.891	104.844	77.647	352.725
23	71.473	245.025	15.88	21.721	201.183	83.683	347.493

Table 3: Members of the Pareto Front generated through Aggregated LHS (Long Run)

Total Generations Computed	425
Average Time Per Generation (sec)	20.4
Total Wall Clock Time (sec)	8.66E+03

Table 4: Solution Statistics for Aggregated LHS (Long Run)

5.1.2 Aggregate LHS Method

This solution was calculated using parameters designed to produce approximately the same wall clock solution time as the Stochastic Loads Long Run. The command line that was used was:

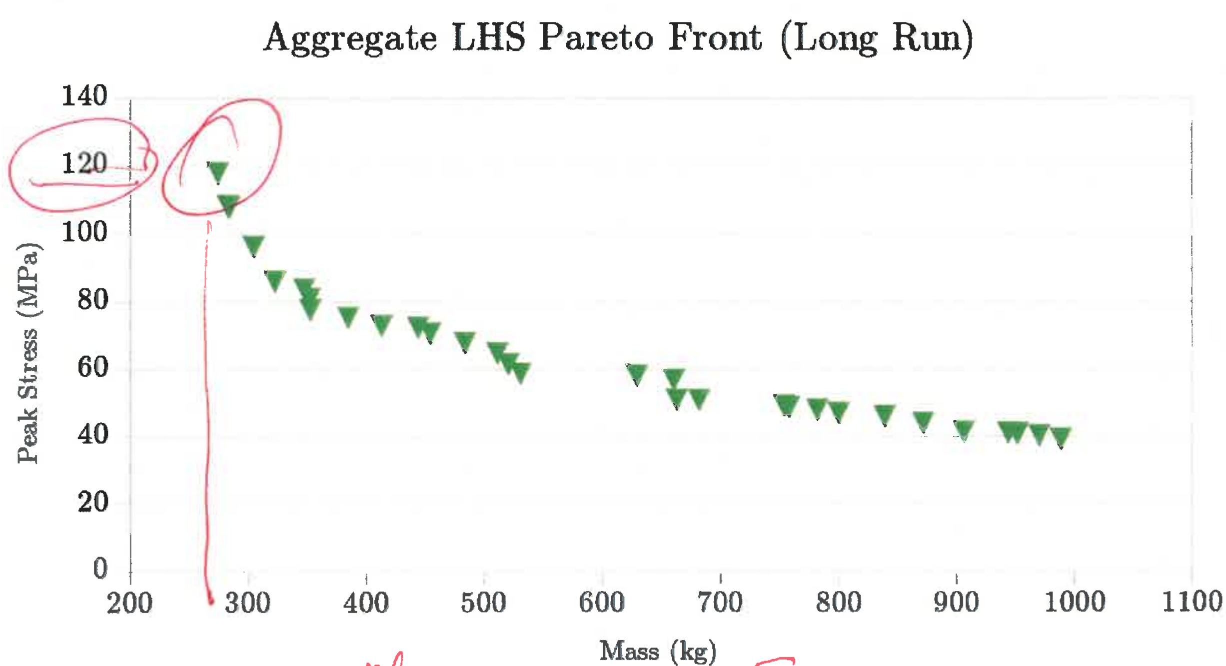
```
python -m pystruct <<data_file>>  -S2 -i80 -g17 --csv
```

Where:

- -S2: Select solution 2: Aggregate LHS method
- -i80: Use a population size of 80.
- -g17: Use a generation count of 17. why? why it is not 200?
- --csv: Generate CSV output for final Pareto Front

Resultant Pareto Front

Table 3 shows the design parameters of the members of the Pareto Front generated by this solution run. The Pareto Front is also given graphically in Figure 7.



are they the same, fig. 7 compared to figure 10

Figure 7: Graph of the Pareto Front generated through Aggregated LHS (Long Run)

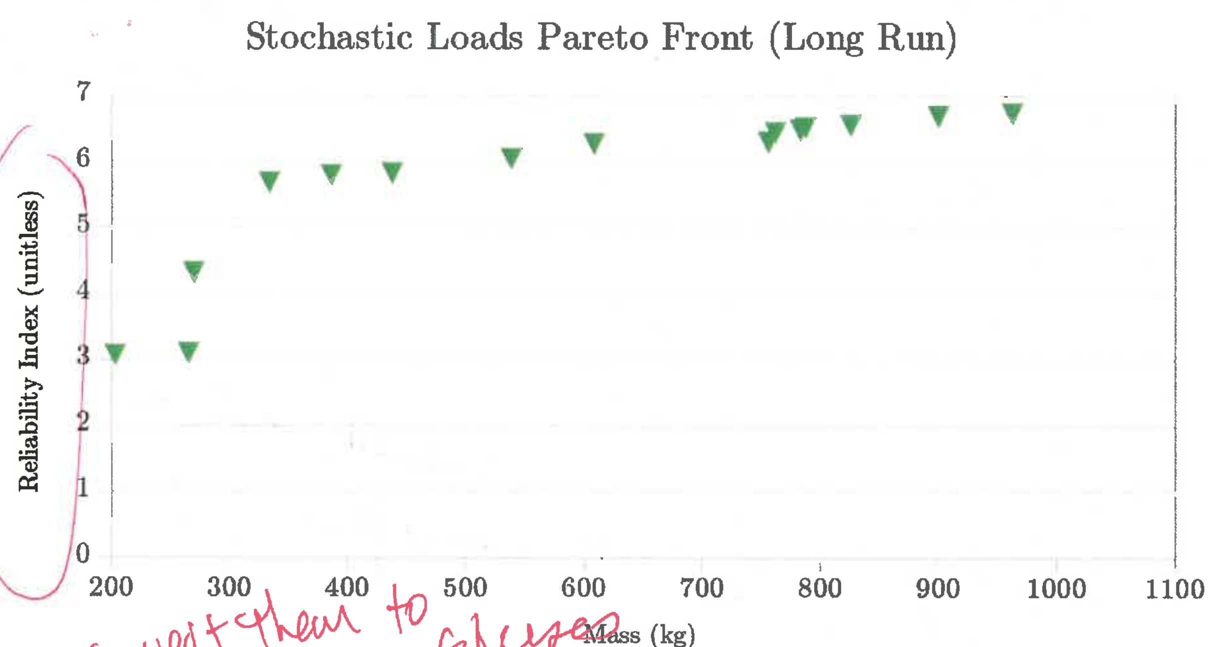
Solution Statistics

This solution also was tracked for several computing performance indicators to compare the performance of the algorithm with the other solutions. These statistics are listed in Table 4.

Design variables | objectives

Top Flange Width	Bottom Flange Width	Web Thickness	Doubler Thickness at Hoist Pin	Doubler Thickness at Load Pin	Reliability Index	Mass
mm	mm	mm	mm	mm	ul	kg
91.997	101.419	6.724	43.826	243.824	3.095	265.219
91.526	236.107	78.99	33.03	89.924	6.481	783.150
80.127	253.897	20.274	19.704	78.611	5.672	333.867
49.012	277.521	86.506	43.241	216.625	6.688	900.125
174.133	194.853	30.888	18.281	107.054	5.815	437.764
35.949	103.641	76.501	32.843	249.291	6.298	756.122
51.253	118.064	7.176	87.576	30.322	3.065	203.018
63.211	102.335	18.535	14.396	127.905	4.316	269.780
132.862	263.599	60.726	106.691	219.739	6.516	787.654
169.514	201.544	73.331	34.801	246.143	6.554	825.922
207.564	215.857	71.357	30.79	87.27	6.439	762.066
89.162	261.263	91.525	48.149	244.42	6.73	962.654
87.785	198.101	44.651	44.423	133.73	6.042	538.436
78.561	223.986	52.554	30.01	171.362	6.268	608.499
72.355	246.809	29.056	20.85	59.901	5.787	386.186

Table 1: Members of the Pareto Front generated through Stochastic Loads (Long Run)



How to convert them to stresses

Figure 6: Graph of the Pareto Front generated through Stochastic Loads (Long Run)

Total Generations Computed	200
Average Time Per Generation (sec)	44.6
Total Wall Clock Time (sec)	8.92E+03

Table 2: Solution Statistics for Stochastic Loads (Long Run)

CHAPTER 5

SELECTED RESULTS

The results for sample runs of each solution strategy are printed below. Each method was run twice. One set of solutions were designed to solve in approximately 2.5 hours on the reference hardware. The other run was targeted at approximately an hour. Each pareto optimal design is summarized and the design parameters given in tabular form. Also given are the solution statistics for each run, which include resource usage and wall clock time taken to arrive at a solution. After each set of results is given, the design parameters found by each strategy is compared, as well as the solution statistics.

→ based upon population size & # of generations

5.1 Long Runs

The following solutions represent example runs of the solvers when given parameters that result in longer run times. These solutions were run with an expected solution time of approximately 2.5 hours.

5.1.1 Stochastic Loads Method

The Stochastic Loads solver was run with the following argument list:

```
python -m pystruct <<data_file>>  -S3 -i80 -g200 --csv
```

Where:

- `-S3`: Select solution 3: Stochastic Loads Method
- `-i80`: Use a population size of 80.
- `-g200`: Use a generation count of 200.
- `--csv`: Generate CSV output for final Pareto Front

should reach at convergence ??

Resultant Pareto Front

Table 1 shows the design parameters of the members of the Pareto Front generated by this solution run. The Pareto Front is also given graphically in Figure 6.

Solution Statistics

This solution also was tracked for several computing performance indicators to compare the performance of the algorithm with the other solutions. These statistics are listed in Table 2.

Parameter	Mean Value	Standard Deviation
Hoist Load X Direction	0N	13kN
Hoist Load Y Direction	150kN	19.5kN

~~Figure~~ 5: Stochastic Parameters for Example System

Table

Analyze Load Case

The actual analysis of the load case selected is nearly identical to the ALHS Approach. The key difference is the use of the unitless parameter β to represent the stress on the beam. This allows the stress, and by inference the safety factor of the device, to be represented in stochastic terms. For a complete discussion and derivation of the parameter β, see section 2.2.2. Using β and the component weight as fitness measurements, MODE is performed on the load case.

Reporting

MODE optimization on the load case defined above generates a single Pareto Front that is reported to the user as recommended designs. Also reported is the Pareto Front presented graphically.

How to rank them?

3. Aggregating the Pareto Fronts from all load cases to find the overall best designs across all load cases.

Each individual step of this process is outlined in detail below.

How to select for each load?

Identify Load Cases

In order to find load cases for use, a Latin Hypercube Sampling algorithm is used to select 1000 independent loading conditions within the set of possible load conditions, which is assumed to be a set of normally distributed sample spaces. In this case, the most conservative loadings are those significantly above the mean for the magnitude of the applied force. In order to ensure reliability, the set of load cases to be analyzed is restricted to those load cases more than 1.96 standard deviations above the mean value of the applied load. This ensures that all of the load cases considered are in the top 2.5% of the sample space. This will result in a set of approximately 25 load cases to consider.

May need describe th with more mathemati cal vigor

Analyze load Cases

For each load case selected, a set of 50 randomly selected candidate designs are generated. These designs are analyzed using Finite Element Analysis to find the maximum stress and design weight. These values are used as fitness functions to rank the designs. This process is repeated using a Multi Objective Differential Evolution algorithm to selectively refine the designs for a set number of generations.

Section needs expanding

Aggregation

At this point, the solution algorithm has 25 separate load cases with individual Pareto fronts. The designs that make up each Pareto front are added to a single unified set of designs. From there, they are once again ranked according to dominance and a "final" Pareto front is generated. The designs that comprise this "final" Pareto front are considered to be the most desirable designs and are presented to the designer.

based upon Eq. (1)

4.2.2 Stochastic Loads Approach

The stochastic loads approach is functionally similar to the Aggregated Latin Hypercube Sampling Approach, with a few key differences:

1. Only a single load case is considered.
2. Stress is not directly considered a fitness variable.

The overall process is detailed below.

Load Case

Instead of using multiple load cases, this approach uses a single load case. However, it is specified as a stochastic set of loads instead of discrete loads applied. For example, the load case considered for the Example System is shown in ~~Figure~~ 5. Note that this information matches the design parameters given in section 1.2.2. Specifying the load case this way effectively covers the entire performance envelope and makes the single load case valid for all states of load on the beam.

Table

CHAPTER 4

SOLUTION METHODS

With all of the principals having been covered previously, the actual solution methodology can be discussed. This chapter provides an overview of the actual solution process used in the code that was developed. While it does not provide a full analysis and discussion of the entire codebase, it should provide sufficient information for the reader to understand the actual code provided.

4.1 Modeling the base beam

For both solution methods, a common basic design of the beam to be studied was constructed. For this study, a basic design using two "C"-style sections to form the two main moment-bearing sections was selected. This style was selected because the cross section lends itself to parameter-based optimization, whereas more complex designs such as box beam spreaders are more well suited to full geometry optimization. Section 1.2.1 has details on the design.

This basic design was modeled in Siemens's Finite Element Pre-processor, FEMAP. The modeled area takes advantage of the two-plane symmetry in the beam to only model one quarter of the beam. It is worth noting that the loading on the beam is affected by the symmetry, and the loads presented for analysis are one fourth of the actual beam loadings.

4.2 Solution Strategies

Two solution strategies were attempted for the example problem proposed. One was termed as the *Aggregated Latin Hypercube Sampling Approach*, while the other was named the *Stochastic Loads Approach.* The following sections describe each solution method in detail and outline similarities and differences between them.

4.2.1 Aggregated Latin Hypercube Approach

In a nutshell, the Aggregated Latin Hypercube (ALHS) Approach analyzes a variety of discrete load cases and collects Pareto Front data for each load case. It does this by:

1. Identifying Load Cases to consider

2. Performing MODE Optimization for each identified load cases. This generates a unique Pareto Front for each load case.

3.2.2 msslhs

msslhs is a Python utility that provides latin hypercube sampling in several locations throughout the code presented. This utility is created and maintained by Justin Hughes. It is available from the Mississippi State University CyberDesign Wiki [6].

CHAPTER 3

TOOLS AND SOFTWARE USED

For this study several externally developed or commercially purchased tools, hardware, and software were used. Each major tool or piece of hardware will be briefly introduced and given a brief description of its source, purpose and method of procurement.

3.1 Computing Hardware

The analyses presented in this report were obtained using commercially available computing hardware. The computer's relevant specifications are given below:

Cite the spec sheet for These parts!

- CPU: AMD Ryzen 2400G. 4 Processing Cores with 8 execution threads. Frequency during tests: 3.8GHz
- Memory: 16GB DDR4 Memory at a frequency of 2666 MHz
- Storage: Intel 540 Series SSD. Capacity:240GB, up to 540 MBps read speed, 490 MBps write

While this computer is a general purpose unit and sees daily use outside the scope of this report, no other tasks were performed simultaneously with the workloads presented herein. Execution times presented represent near-maximum performance for these workloads.

3.2 Software

3.2.1 NASTRAN

The name NASTRAN is short for **NASA Structural Analysis**. It is a finite element solver originally developed by NASA. It enjoys wide popularity throughout industry and sees use for performing linear and nonlinear structural analyses on a variety of materials.

The version of NASTRAN employed in the code presented herein is a recently open-sourced copy of the NASTRAN95 solver. As the name implies, it is a version originally developed in 1995. While newer versions of this software exist, this is the newest copy that is freely available. For simple solutions such as those needed for this work, NASTRAN95 is functionally very similar to the more modern commercial utilities available. This version of NASTRAN is freely available on GitHub.

Whole section needs citations.

Figure 4 shows the named regions as they correspond to portions of the model/drawing.

Also worth noting is that this model is a partial representation of the whole beam. As the example beam in figure 1 shows, the beam features symmetry along the front view. Due to the typical construction of these beams, they also exhibit symmetry along the top view as well. Because of this, the model presented is only one quarter of the complete beam. To account for this, constraints are imposed along the symmetry cut that simulate the remainder of the beam. These constraints can be seen in figure 3 as symbols along the right edge of the diagram. These constraints are along model axes 1, 3, 4, and 5. This equates to constraining X and Z translation, as well as X and Y rotations. This simulates a moment bearing connection at the center of the beam. This is a commonly accepted way of simulating symmetry in NASTRAN. Needs Citation

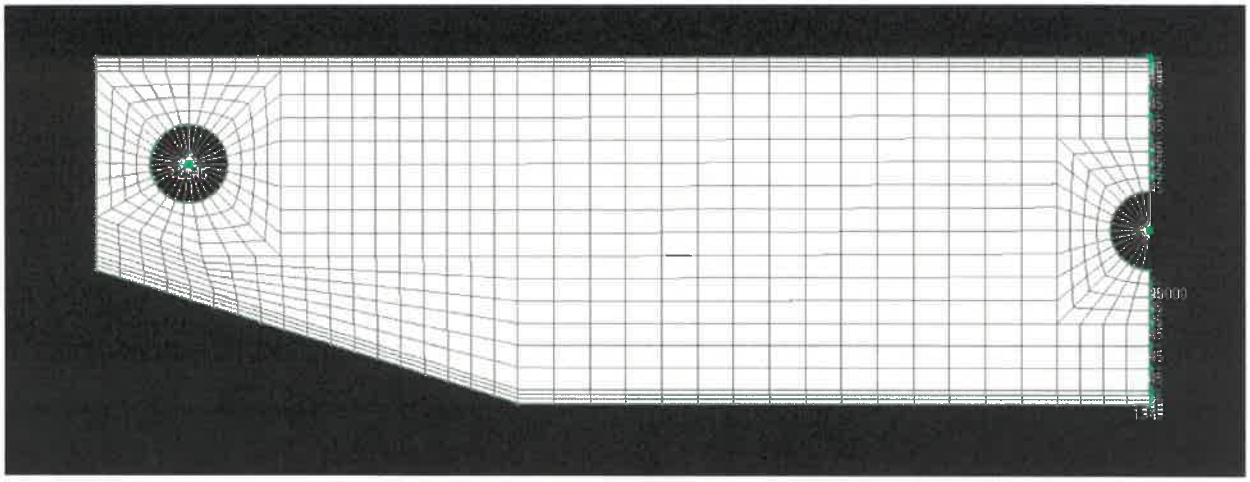


Figure 3: Mesh Geometry as shown in Siemens Femap

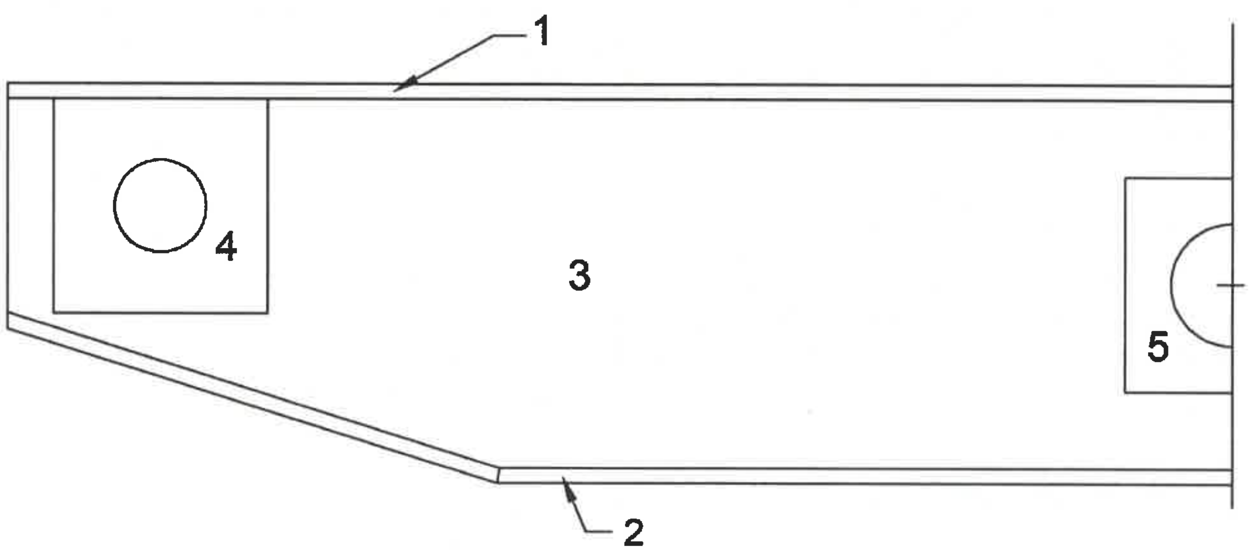


Figure 4: Diagram of modeled beam showing region numbers

construct and solve equations 3 and 4 for σ. It is then easy to substitute the results of equations 3 and 4 into 2 and determine β. In the code presented with this report, all of these terms are assembled separately and combined. Therefore, the final equation with all substitutions performed will not be shown here.

2.3 Finite Element Modeling

When performing optimization to find designs, numerous designs must be analyzed and evaluated for fitness. In some cases, these designs and their fitness functions can be modeled using closed form equations. In the many cases, however, the designs are complex enough that finding closed form models of their behavior is infeasible. In the case that no closed form solutions can be found, numerical analysis can offer an alternative method of evaluating for fitness.

For this study, numerical analysis in the form of Finite Element Modeling was chosen as the method for evaluating the designs for fitness. Each design and load case each are represented as slight modifications to a basic model file containing the fixed geometry and "starting" parameters for the design and loading variables.

yes

Do I need summarize the basics of finite element modeling?

This study makes exclusive use of quadrilateral shell elements, known to NASTRAN as CQUAD4 elements. These elements are 2-dimensional, but do model deformation in all thee dimensions. However, an important assumption made by using these elements is that the entire thickness of the plate distributes the stress applied to the element equally. This implies that deformations in the thickness direction cannot be modeled this way. For example, if a plate can buckle or bend through its thickness this type of element would not be suitable. This model makes use of these elements for the beam flanges, but the elements are oriented to be coplanar with the beam web. This means the flange width is modeled as plate thickness. This orientation will not detect local flange effects such as buckling or local bending. However, this study is primarily concerned with failure of the gross section. Due to the system geometry, these stresses are negligible and therefore the model's inability to properly model them does not appreciably affect the accuracy of this model. This modeling strategy also has the advantage of simplifying the alteration of the flange thickness programmatically.

2.3.1 Selected Model

The model developed for the example problem considered in this work is shown in figure 3. This model was developed using the dimensional data shown in figure 1.2.1. As shown, the model consists of several regions, all of which are modeled using CQUAD4 elements. The regions that are of importance for this study are:

1. The top flange
2. The bottom flange
3. The web
4. A thickened region surrounding the hoist mounting lug
5. A thickened region surrounding the load mounting lug.

multiplying a scalar and a matrix. Therefore, the above equation can be rewritten as:

$$\begin{aligned}
\gamma(\sigma') &= (\sigma_{Px} \cdot P_x + \sigma_{Py} \cdot P_y) - \frac{1}{3}\left(\mathrm{tr}\left(\sigma_{Px} \cdot P_x\right) + \mathrm{tr}\left(\sigma_{Py} \cdot P_y\right)\right)[\mathbf{I}] \\
&= (\sigma_{Px} \cdot P_x + \sigma_{Py} \cdot P_y) - \frac{1}{3}\left(\mathrm{tr}\left(\sigma_{Px} \cdot P_x\right)[\mathbf{I}] + \mathrm{tr}\left(\sigma_{Py} \cdot P_y\right)[\mathbf{I}]\right) \\
&= (\sigma_{Px} \cdot P_x + \sigma_{Py} \cdot P_y) - \frac{1}{3}\left(P_x \cdot \mathrm{tr}\left(\sigma_{Px}\right)[\mathbf{I}] + P_y \cdot \mathrm{tr}\left(\sigma_{Py}\right)[\mathbf{I}]\right) \\
&= P_x \cdot \left(\sigma_{Px} - \frac{1}{3}\mathrm{tr}(\sigma_{Px})[\mathbf{I}]\right) + P_y \cdot \left(\sigma_{Py} - \frac{1}{3}\mathrm{tr}(\sigma_{Py})[\mathbf{I}]\right) \\
&= P_x \cdot \gamma(\sigma_{Px}) + P_y \cdot \gamma(\sigma_{Py}) \qquad (15)
\end{aligned}$$

Note that the terms $\gamma(\sigma_{Px})$ and $\gamma(\sigma_{Py})$ do not contain any terms related to P_x or P_y. This implies that they are constant when deriving with respect to these variables. From this point forward, we will refer to these deviatoric unit tensors as:

$$\begin{aligned}
\gamma(\sigma_{Px}) &= \sigma_{devx} \\
\gamma(\sigma_{Py}) &= \sigma_{devy}
\end{aligned}$$

This means:

$$\sigma_{dev} = \sigma_{devx} \cdot P_x + \sigma_{devy} \cdot P_y \qquad (16)$$

Returning to equation 6, we can begin to define α in terms of these new tensors. To start with, we need to breakdown the ":" operator in the equation. This is the tensor contraction operator, and is defined as:

$$[A] : [B] = \mathrm{tr}([A]\,[B]) \qquad (17)$$

Because both matrix multiplication and the trace of a matrix are both distributive, the tensor contraction operator is also distributive. Using the definition in 17 and applying it to 6 yields:

$$\begin{aligned}
\alpha &= \mathrm{tr}\left[(\sigma_{devx} \cdot P_x + \sigma_{devy} \cdot P_y) \cdot (\sigma_{devx} \cdot P_x + \sigma_{devy} \cdot P_y)\right] \\
&= \mathrm{tr}\left[\sigma_{devx} \cdot \sigma_{devx} \cdot P_x^2 + (\sigma_{devx} \cdot \sigma_{devy} + \sigma_{devy} \cdot \sigma_{devx}) \cdot P_x P_y + \sigma_{devy} \cdot \sigma_{devy} \cdot P_y^2\right] \qquad (18)
\end{aligned}$$

Note that the center term is arranged as shown due to the non-commutative nature of matrix multiplication. Now, we can take this definition of α and easily find the derivatives we need.

$$\frac{\partial \alpha}{\partial P_x} = \mathrm{tr}\left[\sigma_{devx} \cdot \sigma_{devx} \cdot 2P_x + (\sigma_{devx} \cdot \sigma_{devy} + \sigma_{devy} \cdot \sigma_{devx}) \cdot P_y\right] \qquad (19)$$

$$\frac{\partial^2 \alpha}{\partial P_x^2} = \mathrm{tr}\left[2 \cdot \sigma_{devx} \cdot \sigma_{devx}\right] \qquad (20)$$

$$\frac{\partial \alpha}{\partial P_y} = \mathrm{tr}\left[\sigma_{devy} \cdot \sigma_{devy} \cdot 2P_y + (\sigma_{devx} \cdot \sigma_{devy} + \sigma_{devy} \cdot \sigma_{devx}) \cdot P_x\right] \qquad (21)$$

$$\frac{\partial^2 \alpha}{\partial P_x^2} = \mathrm{tr}\left[2 \cdot \sigma_{devy} \cdot \sigma_{devy}\right] \qquad (22)$$

With equations 19 through 22 coupled with equations 8 through 11, we have all of the terms needed to

This turns equation 5 into:

$$\sigma' = \sqrt{\frac{3}{2} \cdot (\alpha)} \tag{7}$$

This simplified equation can be derived as shown below:

$$\frac{\partial \sigma'}{\partial P_x} = \frac{3}{4}\frac{\partial \alpha}{\partial P_x}\left(\frac{3}{2}\alpha\right)^{-\frac{1}{2}} \tag{8}$$

$$\frac{\partial^2 \sigma'}{\partial P_x^2} = \frac{3}{4}\frac{\partial^2 \alpha}{\partial P_x^2}\left(\frac{3}{2}\alpha\right)^{-\frac{1}{2}} - \frac{9}{16}\left(\frac{\partial \alpha}{\partial P_x}\right)^2\left(\frac{3}{2}\alpha\right)^{-\frac{3}{2}} \tag{9}$$

$$\frac{\partial \sigma'}{\partial P_y} = \frac{3}{4}\frac{\partial \alpha}{\partial P_y}\left(\frac{3}{2}\alpha\right)^{-\frac{1}{2}} \tag{10}$$

$$\frac{\partial^2 \sigma'}{\partial P_y^2} = \frac{3}{4}\frac{\partial^2 \alpha}{\partial P_y^2}\left(\frac{3}{2}\alpha\right)^{-\frac{1}{2}} - \frac{9}{16}\left(\frac{\partial \alpha}{\partial P_y}\right)^2\left(\frac{3}{2}\alpha\right)^{-\frac{3}{2}} \tag{11}$$

Of course, this introduces derivatives of α as values that must be calculated. In order to calculate these derivatives, the concept and application of the deviator tensor must be investigated further.

The Deviatoric Stress Tensor (or deviator tensor) describes the component of stress that tends to deform an element. It is given in terms of the overall stress tensor σ' as:

$$\sigma_{dev} = \sigma' - \frac{1}{3}\mathrm{tr}(\sigma')\,[\mathbf{I}] \tag{12}$$

For purposes that will become clear later, we can call this operation a matrix operator γ:

$$\gamma(x) = x - \frac{1}{3}\mathrm{tr}(x)\,[\mathbf{I}] \tag{13}$$

$$\sigma_{dev} = \gamma(\sigma')$$

In this study, σ' is constructed from the component response tensors, σ_{Px} and σ_{Py}:

Explain earlier in the paper how these values come to be

$$\sigma' = \sigma_{Px} \cdot P_x + \sigma_{Py} \cdot P_y$$

Applying equation 13 to the above definition of σ' yields:

$$\gamma(\sigma') = (\sigma_{Px} \cdot P_x + \sigma_{Py} \cdot P_y) - \frac{1}{3}\mathrm{tr}\left(\sigma_{Px} \cdot P_x + \sigma_{Py} \cdot P_y\right)[\mathbf{I}] \tag{14}$$

From here, it is important to remember that taking the trace of a matrix is a distributive operation, as is

Then, one sample is taken from each of the strata. In this implementation, the different variables in the space do not interact, and the given random values of each input variable are combined at random to form a vector of input variables, also known as an individual [7].

2.2.2 The Reliability Index

The reliability index is a unitless constant that can be used to infer the probability of a system or component to fail in service. Roughly, the Reliability index describes the safety factor of a system as the Z-score of a standard normal distribution, where higher numbers indicate higher probabilities of success. The Reliability Index of a loading condition can be written as:

$$\beta = \frac{\mu_{Sy} - \mu_\sigma}{\sqrt{\sigma_\sigma^2 + \sigma_{Sy}^2}} \tag{2}$$

relation to reliability

Where

μ_{Sy} = Mean of the material yielding stress

σ_{Sy} = Std. Deviation of the yielding stress

μ_σ = Mean of the Von Mises stress resulting from the applied load

σ_σ = Std. Deviation of the Von Mises stress resulting from the applied load

The first two variables in the list above relate to properties of the material the object is made from, and are therefore given or assumed. The second two, however, relate to the von Mises stress and must be calculated.

Finding μ_σ and σ_σ

If $g(x_1, x_2, ...x_n) = Y$ denotes a function of multiple random variates and a single, dependent variable, let: cite

$$\mu(Y) = g(x_1, x_2, ...x_n) + \frac{1}{2}\sum_{i=1}^{n}\left(\frac{\partial^2 g}{\partial x_i^2}\sigma_{xi}^2\right) \tag{3}$$

$$\sigma^2(Y) = \sum_{i=1}^{n}\left(\frac{\partial g}{\partial x_i}\sigma_{xi}\right)^2 + \frac{1}{4}\sum_{i=1}^{n}\left(\frac{\partial^2 g}{\partial x_i^2}\sigma_{xi}^2\right)^2 \tag{4}$$

The tensor definition of the von Mises stress at a given location can be written as:

$$\sigma = \sqrt{\frac{3}{2} \cdot (\sigma_{dev} : \sigma_{dev})} \tag{5}$$

Where σ_{dev} is the stress deviator tensor, which will be more completely addressed later. We can simplify derivation of equation 5 by hiding the tensor contraction in the equation with a placeholder variable:

$$\alpha = (\sigma_{dev} : \sigma_{dev}) \tag{6}$$

Then, an operator $P_{a,b}$ can be defined which returns true if C_a dominates C_b. This would take the following form:

$$P_{a,b} = \begin{cases} \text{True when } C_a^i < C_b^i \; \forall i \in \{1..n\} \\ \text{False otherwise} \end{cases} \tag{1}$$

Note that the operator $P_{a,b}$ does not necessarily imply the value of $P_{b,a}$. While only C_b or C_a can be dominant, it is possible that neither is dominant. In this case, both $P_{a,b}$ and $P_{b,a}$ would be false[5].

Implementing Pareto Dominance in DE

In order to extend DE to be used with multi-objective problems, the code presented in this report simply made a slight modification to the selection operator shown in code listing 3:

```
output_vector = empty_2d_vector[num_individuals][num_design_vars]
for integer j in [0 ... num_individuals]:
    //These statements would return arrays of several fitness values.
    f1 = get_fitness(parent_vector[j])
    f2 = get_fitness(child_vector[j])

    if P(f2,f1)
        output_vector[j] = child_vector[j]
    else
        output_vector[j] = child_vector[j]    ? parent_vector
return child_vector
```

Listing 4: Modified Selection Operator

Note that the structure of the if condition "defaults" to selecting the parent. The child is only selected if it is dominant. The parent is selected if it is dominant or if neither dominates.

2.2 Random Loads

This study presents two different approaches for performing MODE optimization on finite element models. The key difference in how the two methods work is the choice of how the random loads in the problem are handled. In this section, the two strategies for generating random loads are introduced.

2.2.1 Latin Hypercube Sampling

Latin hypercube sampling is used throughout the code presented here to generate evenly distributed random data. this method of sampling was selected due to its ability to evenly spread out the sampling points throughout the variable's domain. It does this by programmatically ensuring that each value of each input variable is only represented once. It does this by separating the domain of each input variable into a predetermined number of strata, each with an identical probability of containing an arbitrary sample point.

```
return child_vector
```

Listing 2: Pseudo-code for the Crossover Operator [5]

Selection

The final major phase in the Differential Evolution loop is the Selection Operator. This operator takes the parent and child vectors and calculates the value of the objective function for each individual. The operator then compares each parent individual's fitness against the associated child individual. The one with a more desirable fitness value is retained and added to the *output vector* while the other is discarded. The general process is outlined below [5]:

```
output_vector = empty_2d_vector[num_individuals][num_design_vars]
for integer j in [0 ... num_individuals]:
    f1 = get_fitness(parent_vector[j])
    f2 = get_fitness(child_vector[j])
    if f1 < f2
        output_vector[j] = parent_vector[j]
    else
        output_vector[j] = child_vector[j]
return child_vector
```

Listing 3: Pseudocode for the Selection Operator (Assuming a minimization approach) [5]

How ~~to~~ determine the convergence? when to stop the iteration

2.1.2 Extending DE to Multi-Objective

Differential Evolution Optimization is originally a single-objective method. However, it can easily be extended to multi-objective operation by changing the method by which fitness is evaluated. Instead of a single fitness function, multiple independent fitness functions are evaluated using the concept of Pareto dominance.

Complete this section

Pareto Dominance

Pareto Dominance is a simple way to compare systems based on multiple different fitness criteria. Pareto dominance for a minimization problem can be described by the following formula [5]:

Let the vector of fitness values for two arbitrary solutions be defined as:

$$C_a = \{f_1, f_2, ...f_n\}$$
$$C_b = \{F_1, F_2, ...F_n\}$$

Where:

$f_{1..n}$	:	Fitness values for C_a
$F_{1..n}$	:	Fitness values for C_b
n	:	Number of fitness values per design (number of objective functions).

Mutation

The mutation operator is the first step for each cycle of the optimization loop. This algorithm emulates random mutations in genetic code commonly found in nature. Taking each individual from the vector of parents, a mutated vector of properties is generated. These vectors are known as *trial vectors.* To perform this action, the basic process flow shown below is employed [5]:

```
trial_vector = empty_2d_vector[num_individuals][num_design_vars]
for integer j in [0 ... num_individuals]:
    x_1 = individuals[j]
    x_2 = random_member_of_individuals
    x_3 = random_member_of_individuals
    beta = x.x // (arbitrary amplification factor selected by user)
    for integer i in [0 ... num_design_vars]:
          trial_vector[j][i] = x_1[i] + beta * (x_2[i] - x_3[i])
return trial_vector
```

Listing 1: Pseudo-code for the Mutation Operator [5]

This set of trial vectors is one of the distinguishing facets of Differential Evolution. If the equation on line 8 above is reviewed carefully, it can be seen that the trial vector is different than its associated parent vector by the distance between 2 other random individuals within the solution space. This has the interesting effect of causing the differences to between parent and trial vectors to change based on the condition of the solution. Early in the solution process when the individuals are sparsely spaced across the solution space, the trial individuals tend to spread apart similarly. In later cycles as minima start to become identified, the distance between individuals becomes smaller. This has the effect of making the trial vectors land more closely to their associated parents. This allows Differential evolution to converge relatively quickly once minima start to appear in the solution space [5].

Crossover

The crossover operator combines the parent individuals and their associated trial individuals to make a single set of *child individuals.* It does this using the following general procedure [5]:

```
child_vector = empty_2d_vector[num_individuals][num_design_vars]
for integer j in [0 ... num_individuals]:
    C = x.x // Constant that dictates how often the crossover picks from the
          // trial vector.
    for integer i in [0 ... num_design_vars]:
        rnd = make_random_number()
        if rnd > C
            child_vector[j][i] = trial_vector[j][i]
        else
            child_vector[j][i] = parent_vector[j][i]
```

CHAPTER 2

SOLUTION PRINCIPALS

The solution code in this study relies on several principals that are complex enough to require seperate attention prior to introducing the solution method itself. This chapter provides a brief introduction of selected major concepts used in the solution. Each concept is related to the actual solution in general terms.

2.1 Numerical Optimization

Optimization is defined as "a mathematical technique for finding a maximum or minimum value of a function of several variables subject to a set of constraints." [4] In the specific case of engineering design, one of several techniques is used to find local or global extrema of a function of one or multiple variables. These techniques use various criteria to traverse the independent variables and detect these extrema. The method the optimizer uses to traverse the solution space has a significant impact on the speed at which the operation converges to a solution or solutions.[3]

Is this section complete enough?

2.1.1 Differential Evolution

Introduce the concep of the obje tive functi

Differential evolution was selected as the optimizer for this study.

Differential Evolution is a member of optimizers collectively known as *Evolutionary Algorithms*. These optimizers use various approaches to emulate the concept of biological evolution to optimize a given objective function. Differential optimization performs this task through the use of a 4-phased approach: Initialization, Mutation, Crossover, and Selection. [5]

Initialization

Prior to starting the optimization loop, an initial "population" of individuals has to be generated for the optimizer to work from. An individual consists of a vector $\vec{x}$ of values of the objective function's independent variables. The Initialization process generates a vector $\vec{X} = \{\vec{x}_1, \vec{x}_2 \cdots \vec{x}_n\}$ of individuals by randomly selecting values for the independent variables. This vector represents the complete population that will be operated on in the first generation of the optimization loop. These individuals will be collectively be known as *parents*.[5]

In the case of the implementation presented here, Latin Hypercube Sampling (LHS) was employed to generate the random values to assemble $\vec{X}$. More detail on LHS can be found in section 2.2.1.

1. Minimize component mass, expressed as the beam structure's mass in kilograms.
2. Minimize stress in the beam, through one of two criteria depending on which approach is being used:
   (a) Directly minimize the peak stress in the beam, or
   (b) Maximize the critical value for the constant β, which is discussed further in section 2.2.2 on page 13.

relation to reliability

1.2.4 Constraints

During the MODE optimization process, 1 constraint is applied. The constraint requires the mass of the modeled beam portion to remain under 1 metric ton (1000kg). In order to accomplish this, a fitness penalty method is used which triggers if the inequality:

$$W_{beam} \leq 1000kg$$

is violated.

The above constraint is applied by generating a multiplier based on the degree to which the constraint has been violated. This multiplier is used to multiply the fitness results, ensuring the solutions that violate constraints are the least dominant and are therefore much less likely to be selected to move on to the next generation in favor of more dominant designs.

1.2.5 design variables?

which dimensions can be changed

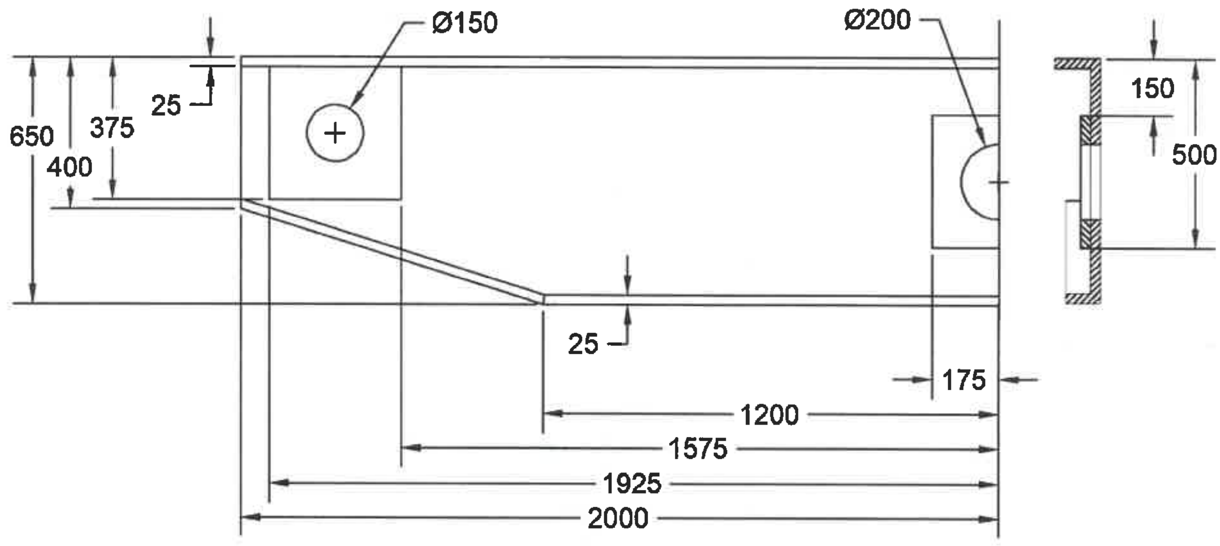


Figure 2: Fixed Dimensions for the Example System Beam

Also fixed is the material, which is assumed to be ASTM A36 steel. The material has properties assumed to be:

- Yield Strength: Mean 250MPa, Std. Deviation 32.5 MPa
- Young's Modulus: 200 GPa

1.2.2 Performance requirements

In this case, performance requirements primarily relate to the lifting capacity and the allowable side pull on the beam. The requirements selected for this problem are:

1. Lifting capacity (Vertical Maximum Force): 60 metric tons, 60,000kg, 132,000 lbm, 589 kN. For this problem, 600 kN was used. (or)
2. Side Loading Capacity (Horizontal Maximum Force): Simulated 5 degree angle between equalizer beam attachment loads and the vertical axis: 52kN.

The beam to be analyzed is symmetric on 2 planes. This allows for the model to consist of only one quarter of the beam under study. Additionally, It will be later clarified that the selected solution methods utilize stochastic methods to model the loading. To enable the use of stochastic method of solution, the following statistically-defined loads have been used to approximate the above performance requirements:

- Load magnitude, vertical: Mean: 150 kN, Std. Deviation: 19.5 kN
- Load magnitude, horizontal: Mean: 0 kN, Std. Deviation: 13 kN

why they are not the same

1.2.3 Design Objectives

For this design, we seek designs of minimum weight and maximum strength in the loading conditions the part is subject to. Formally stated, the objective functions for the optimized designs are:

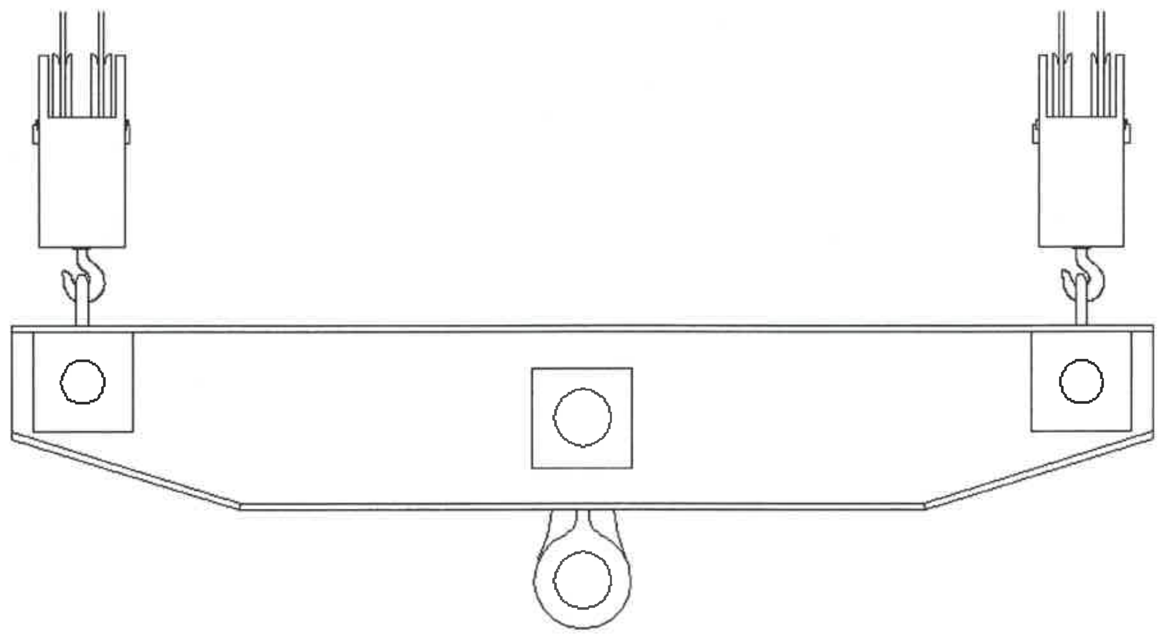


Figure 1: A Basic Equalizer Beam

with a variety of loads and loading orientations. To address this, two strategies have been investigated to attempt to handle load input variation in the design. One makes use of a variant of Monte Carlo simulation, and the other uses reliability centric methods.

1.1 Objective

Mention the random load

This study will seek to develop a method to design an equalizer beam using multi-objective optimization. Furthermore, the study will seek to incorporate treatment of uncertain input loads in the solution method. The development of the solution will focusing on the optimizer design, using relatively simple methods of varying design parameters.

to count on the uncertain input loads

1.2 Problem Description

As mentioned, the aim of this project is to evaluate multi-objective optimization as a design tool for use in the development of designs for equalizer beams. In order to perform this evaluation and provide a "benchmark case" for any comparisons between methods, an example system will be used as a subject for the design. The basic parameters for the solution are presented below.

1.2.1 Example System

The example system to be studied is based on a few basic fixed design parameters. The basic outline of the beam structure is shown in Figure 1.2.1.

CHAPTER 1

MOTIVATION, PROBLEM DESCRIPTION, OBJECTIVE AND EXPECTED OUTCOME

In industry, cranes are commonly used to lift heavy materials and transport them from location to location. In some cases, equipment or material may exceed the lifting capacity of a single crane or lifting fixture. In these cases, multiple cranes must be used to move the load. In order to ensure multi-crane lifts are performed safely, it must be ensured that each crane shares the load equally. Structural components known as equalizer beams are commonly employed to ensure that these loads are evenly distributed between the cranes.

Equalizer beams are typically single weldments with few, if any, moving parts. They are typically constructed from steel. The beam is typically built with 3 major attachment points. One for the load to be lifted, and 2 equidistant crane attachment points. The centers of the three attachment points are typically along the same axis to ensure equal load sharing between the two cranes, even in the event that the beam is out of level. A typical design for an equalizer beam is shown in Figure 1.

Currently, the commonly accepted industry standard in use in the US is ASME BTH-1[2]. This design standard lists recommended minimum design standards for these devices. This standard is widely accepted and is referenced in ASME's safety standard for below the hook devices, B30.20 [1]. The designs prepared and available for purchase in industry tend to be pre-engineered units utilizing standard components and beam sections. In some cases, the performance of these off-the-shelf components does not meet the needs of certain extremely stringent operating environments. Since the weight of rigging equipment is considered part of the lifted load, lifting heavy loads near the capacity of the cranes in use can require extremely weight-efficient rigging. At the same time, safety considerations require the stress in these components to be kept as low as achievable to maximize the margin of safety in the rigging system. Because these two objectives are often at odds with one another, design of these equalizer beams taking both objectives into account can be a difficult exercise for the designer.

The nature of the competing requirements imposed against the design of equalizer beams makes them suitable for multi-objective optimization. In general, multi-objective optimization is intended to find a series of optimal solutions across a range of values for the two (or more) different design objectives and present them to a designer. For example, in the case of the equalizer beam, multi-objective optimization can present a series of optimal designs that have the lowest stress at numerous different weight values. Using this technique, a designer can evaluate a number of designs at numerous different combinations of the design variables and select a design that best suits the situation.

Normally, even multiobjective optimization is implemented to handle situations where the input parameters (such as input loads) are clearly defined. In the case of lifting beams, the components can be presented

List of Code Listings

1 Pseudo-code for the Mutation Operator [5] 10
2 Pseudo-code for the Crossover Operator [5] 10
3 Pseudocode for the Selection Operator (Assuming a minimization approach) [5] 11
4 Modified Selection Operator 12

List of Figures

1 A Basic Equalizer Beam . . . 6
2 Fixed Dimensions for the Example System Beam . . . 7

3 Mesh Geometry as shown in Siemens Femap . . . 17
4 Diagram of modeled beam showing region numbers . . . 17

5 Stochastic Parameters for Example System . . . 23

6 Graph of the Pareto Front generated through Stochastic Loads (Long Run) . . . 25
7 Graph of the Pareto Front generated through Aggregated LHS (Long Run) . . . 26
8 Comparison of the Two Long Run Pareto Plots . . . 28
9 Graph of the Pareto Front generated through Stochastic Loads (Short Run) . . . 29
10 Graph of the Pareto Front generated through Aggregated LHS (Short Run) . . . 31
11 Comparison of the Two Short Run Pareto Plots . . . 32

List of Tables

1 Members of the Pareto Front generated through Stochastic Loads (Long Run) . . . 25
2 Solution Statistics for Stochastic Loads (Long Run) . . . 25
3 Members of the Pareto Front generated through Aggregated LHS (Long Run) . . . 27
4 Solution Statistics for Aggregated LHS (Long Run) . . . 27
5 Members of the Pareto Front generated through Stochastic Loads (Short Run) . . . 29
6 Solution Statistics for Stochastic Loads (Short Run) . . . 30
7 Members of the Pareto Front generated through Aggregated LHS (Short Run) . . . 30
8 Solution Statistics for Aggregated LHS (Short Run) . . . 31

5.1.3 Comparison 28
5.2 Short Runs 28
5.2.1 Stochastic Loads Method 28
5.2.2 Aggregate LHS Method 30
5.2.3 Comparison 31

6 Concluding Remarks **33**

7 References **34**

8 Appendix A: Complete Code Listing **35**

Contents

1 Motivation, Problem Description, Objective and Expected Outcome **5**
1.1 Objective 6
1.2 Problem Description 6
1.2.1 Example System 6
1.2.2 Performance requirements 7
1.2.3 Design Objectives 7
1.2.4 Constraints 8

2 Solution Principals **9**
2.1 Numerical Optimization 9
2.1.1 Differential Evolution 9
2.1.2 Extending DE to Multi-Objective 11
2.2 Random Loads 12
2.2.1 Latin Hypercube Sampling 12
2.2.2 The Reliability Index 13
2.3 Finite Element Modeling 16
2.3.1 Selected Model 16

3 Tools and Software Used **19**
3.1 Computing Hardware 19
3.2 Software 19
3.2.1 NASTRAN 19
3.2.2 msslhs 20

4 Solution Methods **21**
4.1 Modeling the base beam 21
4.2 Solution Strategies 21
4.2.1 Aggregated Latin Hypercube Approach 21
4.2.2 Stochastic Loads Approach 22

5 Selected Results **24**
5.1 Long Runs 24
5.1.1 Stochastic Loads Method 24
5.1.2 Aggregate LHS Method 26

DIFFERENTIALLY EVOLVED DESIGN OF AN EQUALIZER BEAM FOR USE IN GENERAL PURPOSE LIFTING AND HANDLING

Aaron T. Moore
B.S. May 2013, Old Dominion University

A Project Report Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

MECHANICAL ENGINEERING

May 2019

Mechanical and Aerospace Engineering
Old Dominion University
Norfolk, Virginia, USA
2019-04-30

Approved by:
NOTE: This is a draft.
Approvals have not been given.
Dr. Gene Hou (Advisor)