Design of Engine Gasket Using Shape Optimization
EGM6352: Advanced FEM Project 1

Charles Jekel cj@jekel.me

March 30, 2018

Abstract

This report details the design of a rubber gasket utilizing shape optimization. The physical behavior of closing an engine block onto a rubber gasket is modeled using non-linear finite element (FE) analysis. The details of the FE model are described in full. Mooney-Rivlin and Ogden hyperelastic materials models are considered. The shape optimization procedure is described, which modifies the geometry of the rubber gasket. The goal of the optimization routine is to find the geometry of a rubber gasket that produces the minimum closure gap while making the contact pressure as uniform as possible. The optimization problem was solved using a global optimization algorithm that utilizes the Lipschitz function. An optimum design was found that reduced the closure gap and had a more uniform contact pressure.

1 Introduction

The purpose of this project is to design an engine gasket. The gasket aids to prevent engine oil from leaking when the head is sealed. The initial design of the gasket is presented in Figure 1. This design is to be altered in a shape optimization to find the design that reduces gasket closure gap, while producing a uniform contact pressure. A deformed gasket is shown in Figure 2, where the closure gap to be reduced is shown as d. Non-linear finite element (FE) analysis is used to simulate the closure of the gasket. The FE model will aid in the design of an improved gasket.

The project is broken into three parts. Section 2 will cover the material parameter identification, where hyperelastic models are fit to rubber test data. Section 3 will describe the non-linear FE analysis for the initial geometry of the gasket. The details of the FE model will include: element type, integration scheme, mesh size, and a discussion of the results. Section 4 will describe the details of a shape optimization to find the shape of the gasket that reduces the gap while making the contact pressure as uniform as possible.

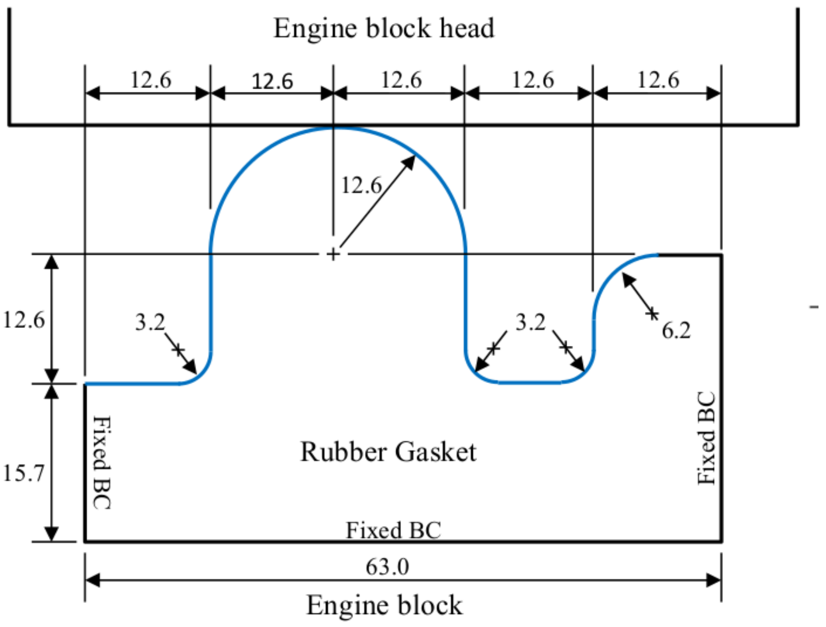


Figure 1: Geometry of the initial rubber engine gasket.

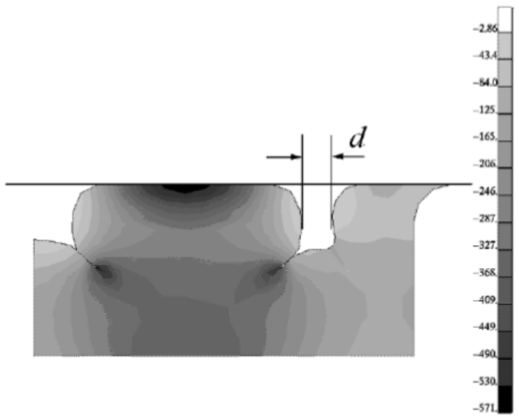


Figure 2: Deformed gasket where the gap is shown as the variable d.

2 Material parameters

Hyperelastic material models, including the Mooney-Rivlin and Ogden models have been commonly used model rubber like materials [1] [2]. The test data available for the rubber gasket material is presented in Table 1. Unfortunately there is no volumetric test data available, so we'll assume that the rubber has a Poisson's ratio of 0.495. This assumption assumes the rubber is nearly incompressible. Material models will be fit to Mooney-Rivlin and Ogden models. The resulting models will be compared, and the best set of parameters identified.

Strain	**Simple tension** (MPa)	**Planar shear** (MPa)	**Biaxial tension** (MPa)
0.0	0.0	0.0	0.0
0.1	4.2	5.0	6.4
0.2	6.4	7.6	9.4
0.3	8.0	9.0	11.0
0.4	9.2	10.2	13.0
0.5	10.0	11.2	15.0
0.6	11.2	12.4	17.6
0.7	12.6	14.8	21.4

Table 1: Available test data for the rubber gasket.

Assuming nominal strains are measured, then the principle stretches can be calculated by

$$\lambda_i = 1 + \varepsilon_i \tag{1}$$

A Young's modulus can be approximated by dividing the tensile stress by the strain yielding:

$$\begin{aligned} E &= \frac{\sigma}{\varepsilon} \qquad (2)\\ &= [4.2, 3.2, 2.67, 2.3, 2.0, 1.87, 1.8] \times 10^8 (\text{Pa}) \qquad (3) \end{aligned}$$

We can see that the Young's modulus varies from 420 MPa to 180 MPa throughout the strain range. The average of these Young's modlus is approximately 258 MPa. This Young's modulus can then be used to approximate the bulk modulus as

$$K = \frac{E}{3(1-2\nu)} \tag{4}$$

where K is the bulk modulus, E is the approximate Young's modulus, and ν is the provided Poisson's ratio. The result yields a bulk modlus of 8.59 GPa.

$$\begin{aligned} K &= \frac{2.58 \times 10^8}{3(1-2(0.495))} \qquad (5)\\ &= 8.59 \text{ GPa} \qquad (6) \end{aligned}$$

2.1 Mooney-Rivlin

The regression matrix of the Mooney-Rivlin model can be assembled in parts. For the simple tension we have that

$$\begin{bmatrix} 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-2}) & 2(1 - \boldsymbol{\lambda}^{-3}) \end{bmatrix} \begin{bmatrix} A_{10} \\ A_{01} \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_u \end{bmatrix} \tag{7}$$

where $\boldsymbol{\lambda}$ is the vector of stretch values, and $\boldsymbol{T}_u$ is the vector of uniaxial stress values. Parameters are determined for the planar sheer test using

$$\begin{bmatrix} 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-3}) & 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-3}) \end{bmatrix} \begin{bmatrix} A_{10} \\ A_{01} \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_s \end{bmatrix} \tag{8}$$

where $\boldsymbol{\lambda}$ is the vector of stretch values, and $\boldsymbol{T}_s$ is the vector of planar shear stress values. Lastly, the biaxial parameters are determined using

$$\begin{bmatrix} 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-5}) & 2(\boldsymbol{\lambda}^3 - \boldsymbol{\lambda}^{-3}) \end{bmatrix} \begin{bmatrix} A_{10} \\ A_{01} \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_b \end{bmatrix} \tag{9}$$

where $\boldsymbol{\lambda}$ is the vector of stretch values, and $\boldsymbol{T}_s$ is the vector of planar shear stress values. Putting this all together, we have a regression matrix $\boldsymbol{X}$ that is the shape 24×2. The regression matrix is defined as

$$\boldsymbol{X}\boldsymbol{A} = \boldsymbol{T} \tag{10}$$

$$\begin{bmatrix} 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-2}) & 2(1 - \boldsymbol{\lambda}^{-3}) \\ 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-3}) & 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-3}) \\ 2(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{-5}) & 2(\boldsymbol{\lambda}^3 - \boldsymbol{\lambda}^{-3}) \end{bmatrix} \begin{bmatrix} A_{10} \\ A_{01} \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_u \\ \boldsymbol{T}_s \\ \boldsymbol{T}_b \end{bmatrix} \tag{11}$$

This equation can be solved in a least squares formulation. There is no need to include the 0.0 strain and stress values, because this formulation ensures that the stress strain data will go through the origin (0,0). Thus there will be a zero residual at this point, and it doesn't effect the least squares fit. Thus we can reduce the 24×2 regression matrix to a 21×2 matrix. The rational for this deriviation is explained by [3].

The results of the lest-squares fit and Abaqus model are shown in Table 2. What is particularly interesting is that the Abaqus model produces different parameters than the least-squares fit. It was expected that the two methods would produce the same parameters. To be sure that no error was performed, the Mooney-Rivlin parameters from Abaqus were plugged into the regression matrix to evaluate the nominal stress values. This resulted in the exact same nominal stress values as Abaqus calculated. It is suspected that Abaqus provided slightly different parameters due to some numerical stability, especially since numerical stability with the bulk modulus k is known to be an issue. While the Mooney-Rivlin parameters are slightly different, they produce a comparable nominal stress values. This will be further discussed in section 2.3, as the Mooney-Rivlin models will be compared to the Ogden models.

2.2 Ogden

Ogden models of $N = 1$, $N = 2$, and $N = 3$ were fit to the test data using Abaqus. Abaqus calculates the parameter coefficients, nominal stress values,

Method	A_{10}	A_{01}	K
Least-squares	4.39 MPa	0.64 MPa	8.59 GPa
Abaqus	4.58 MPa	0.55 MPa	0.51 GPa

Table 2: Mooney-Rivlin material parameters from least-squares fit and Abaqus model.

and the stability of each model. The $N = 3$ demonstrated some instability conditions, so results from this model were disregarded. The $N = 1$ Ogden model has three parameters, like the Mooney-Rivlin model. While the $N = 2$ Ogden model has five parameters. Overall there were minimal differences between the $N = 1$ and $N = 2$ Ogden models, and both models look very similar. The differences between Ogden models and Mooney-Rivlin models will be discussed in section 2.3.

2.3 Comparisons between models

The nominal stress values vs the principal stretch values are plotted for all three test cases. The results for simple tension are shown in Figure 3. The results for planar shear are shown in Figure 4. The results for biaxial tension are shown in Figure 5. The most important distinction is that all models to match the test data exactly. This means that there is no perfect material model to replicate the test data. In general the Ogden models were very similar to each other. The same can be said about the Mooney-Rivlin models.

In order to quantitatively compare the material models, the sum-of-square of the residuals of each model is evaluated based on

$$\boldsymbol{r} = \boldsymbol{T} - \boldsymbol{X}\boldsymbol{A} \tag{12}$$

$$e = \boldsymbol{r}^{\mathrm{T}} \cdot \boldsymbol{r} \tag{13}$$

where e is the sum-of-square of the residuals. The values are shown in Table 3. We can see that the two Mooney-Rivlin models have lower sum-of-squares values than the Ogden models. Thus, we can say that the Mooney-Rivlin models match the test data better than the Ogden models.

The two Mooney-Rivlin models appear to have similar sum-of-squares values, and produce similar stretch-stress curves to each other. Assuming that Abaqus selected parameters that are more stable, as evident by the smaller k, it makes sense to use the Abaqus Mooney-Rivlin model in this

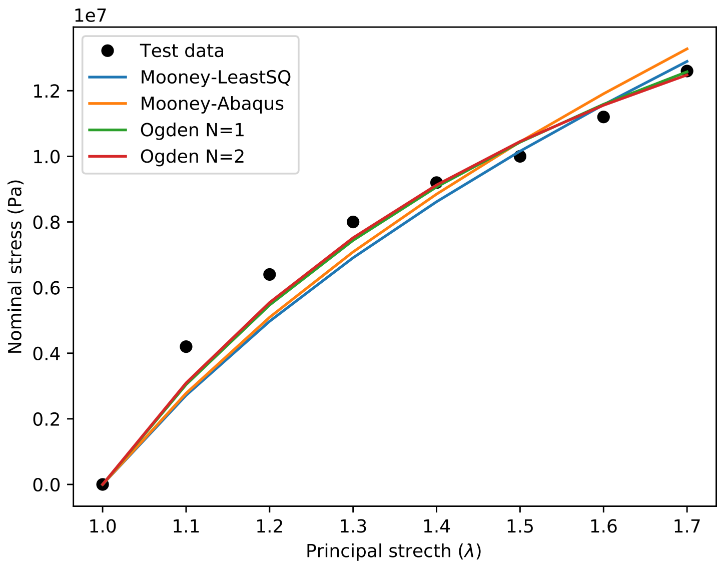


Figure 3: Simple tension results for material models and test data.

Method	**Sum-of-squares** (Pa^2)
Mooney least squares	1.86×10^{13}
Mooney Abaqus	1.95×10^{13}
Ogden $N = 1$	2.65×10^{13}
Ogden $N = 2$	3.19×10^{13}

Table 3: The sum-of-squares of the residuals (e) for each material model.

gasket problem over the least-squares model. However, either model should produce similar results, since the stretch-stress curves were very similar.

The code to perform the least-squares fit, and calculate the sum-of-squares fore each method is presented in Appendix A.

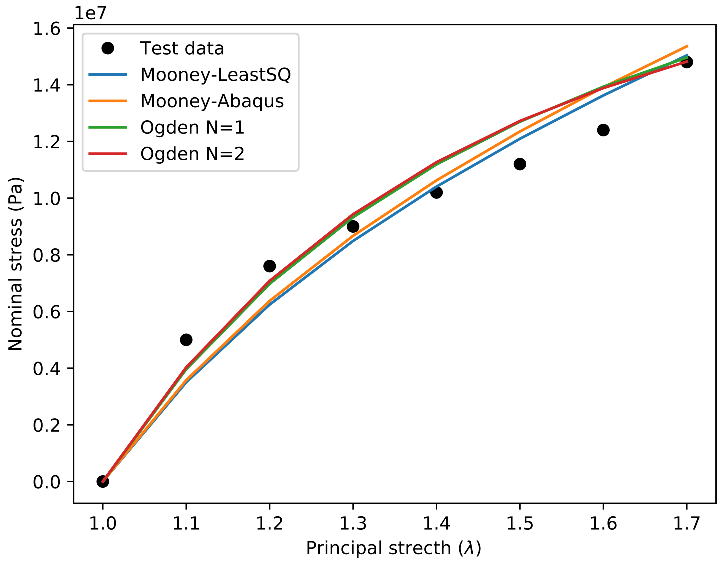


Figure 4: Planar shear results for material models and test data.

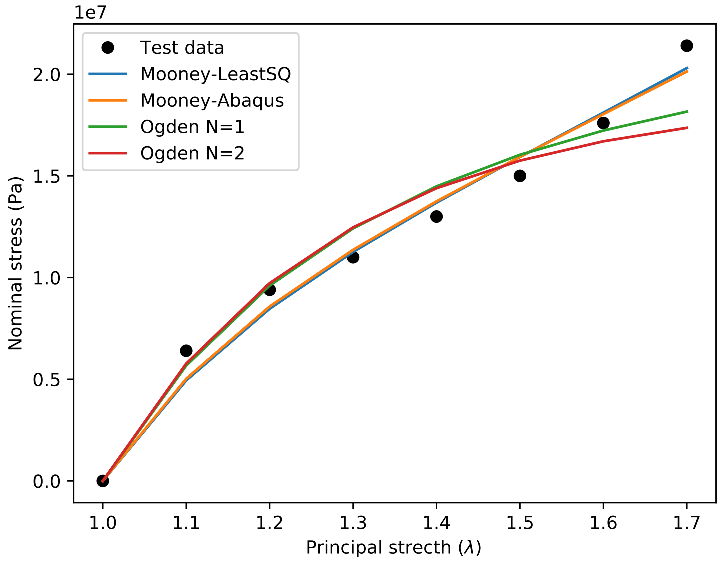


Figure 5: Biaxial tension results for material models and test data.

3 Non-linear FEA

A non-linear finite element (FE) model is constructed to replicate the closing of an engine gasket. The Rubber gasket is modeled as a hyperelastic deformable, while the engine block head is modeled as a rigid body line. Contact is used to model the simulation of the closing engine head. The engine head is closed onto the gasket using a displacement controlled boundary condition on the engine head. The Abaqus input file for a coarse mesh is provided in Appendix B, such that the complete model can be reproduced. The following sections detail finding an appropriate mesh size, discussion of the boundary conditions, and a presentation of the results.

3.1 Mesh

A linear quad-dominate element type with regular quad elements was chosen to model the gasket. Linear elements are chosen rather than quadratic elements due to the application requiring contact between the gasket and engine block. The contact pressure of quadratic elements is somewhat superficial, as there will be positive and negative pressures on the same surface. However, with linear elements the contact pressure will be more uniform.

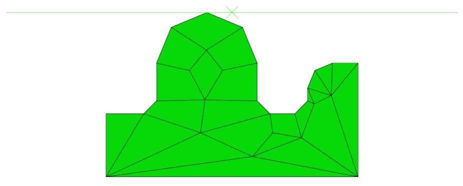
Reduced integration elements and incompatible mode elements generally are known for their resistance to volumetric locking. Volumetric locking is a serious problem for this compression application. However, it appeared that both reduced integration and incompatible mode elements were severely unstable when performing a convergence study. If the mesh was too small, the non-linear FE model would fail to converge. Additionally if the mesh was too fine, the FE model would fail to converge. If these element formulations are absolutely required, this application may perhaps benefit from a re-meshing scheme, in which the deformed mesh would be quality checked at various iterations of the non-linear run. The regular quad elements were chosen because the numerical model was more likely to complete for a variety of meshes. This will become a bigger issue in Section 3 when the shape optimization is performed, as the optimization will require a numerically stable model. While re-meshing could be an interesting approach to deal with the mesh problems, it was considered outside the scope of this work as a re-meshing or mesh-less study could be a project in itself.

The choice of element means that it isn't possible to truly represent a curved surface. This may mean that a larger number of elements are required to obtain a reasonable approximation of the curved surface, as the curved surface is linearly discretized. It is the hope that the the curvature can be approximated by many linear elements.

A mesh convergence study was performed by varying different mesh seeds, and using the Abaqus free quad-dominated automatic mesher. It appeared that the quad-dominated mesh was more resilient to the large deformations than the quad-only mesh. Using the global mesh seed, a nominal element size was specified. The results of the mesh convergence study are presented in Table 4, which presents the maximum von Mises stress and total load. Von Mises stress is not useful as a convergence criteria in this case as there are stress concentrations that excite around the fillets of the structure. The total load was calculated from the reaction force in the vertical direction from the applied contact member. The variation in element size can be seen in Figure 6.

Nominal element (mm)	**Number of elements**	**Maximum von Mises stess** (MPa)	**Total load** (KN)
50	24	36.6	406.5
7.5	71	18.1	350.2
5	117	17.6	359.6
2.5	345	17.3	359.8
1.25	1,352	28.7	355.4
0.70	4,120	40.1	355.8
0.4	12,962	64.1	354.0

Table 4: Results of mesh convergence study.


(a) 50 mm nominal element

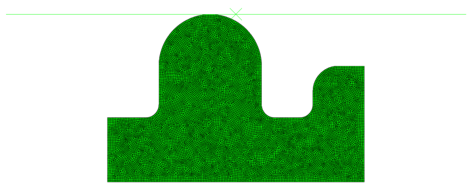

(b) 0.4 mm nominal element

Figure 6: Variation in mesh convergence study shown from the most coarse mesh used to the most fine mesh.

Convergence studies are best visualized on a log-scale, as shown in Figure 7. We can see convergence behavior occurs in the total load at about 1,000 elements. At this point, diminishing marginal returns occur. This means that the computational cost drastically increases while only marginally improving the total load value. Thus for this study it was decided that the nominal element size of 1.25 mm (or 1,352 elements) produce a reasonable total load result. This nominal element size will be presented in the results subsection, and used in the shape optimization of section 3.

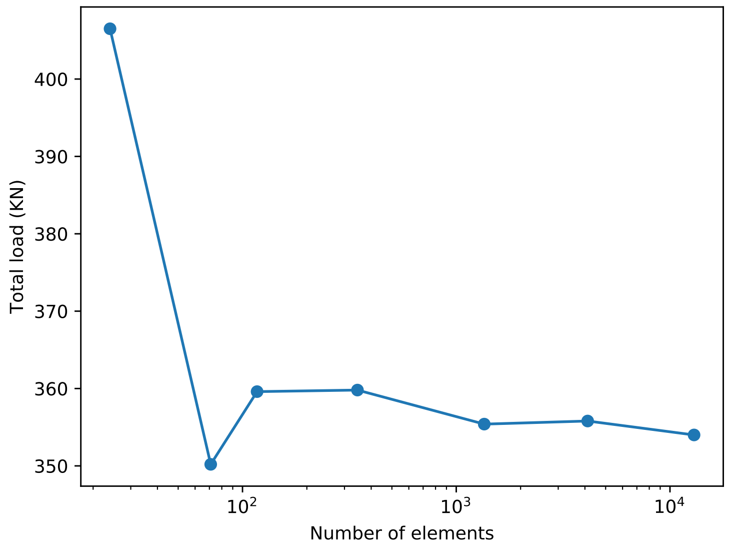


Figure 7: Mesh convergence demonstrated by total load plotted vs the number of elements.

3.2 Boundary conditions

The boundary conditions are best illustrated in Figure 8. The engine block head is modeled as a rigid line, which has a displacement controlled boundary condition which moves down 12.6 mm onto the gasket. This block head is not allowed to move horizontally. The Fixed boundary conditions on the gasket are applied with a pin join on each node. This reduces the need for restricting a rotational degree of freedom, as the two pins prevents rigid body rotation.

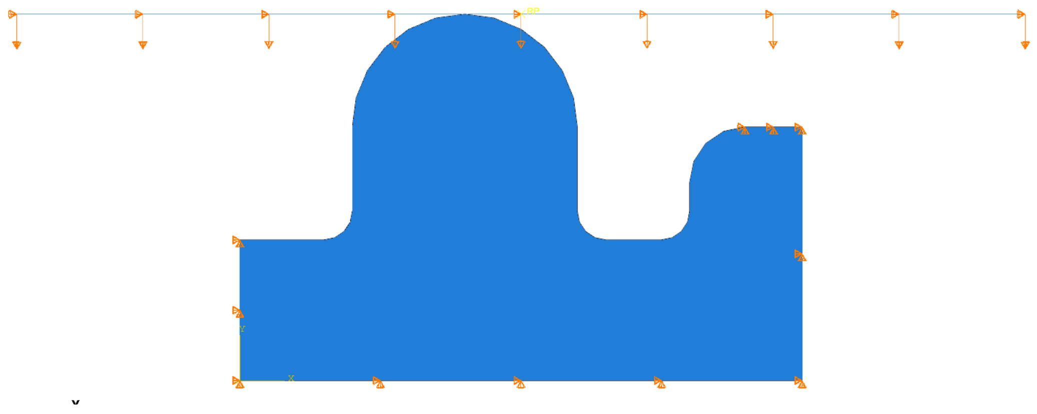
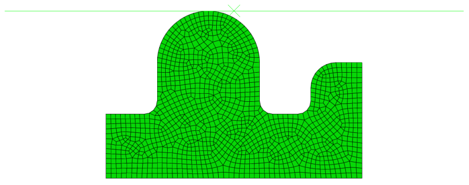


Figure 8: Boundary conditions as applied in Abaqus.

Contact is the primary driver of the applied load. There are two types of contact used: 1) The contact between a rigid body (Engine block head) and a deformable body (Rubber gasket). 2) The self contact of the deformable body (rubber gasket). Both contact properties utilize a tangential behavior card with a friction coefficient of 0.05. The master-slave relationship was used to control the contact interaction. For the self-contact, the master and slave is the surface of the rubber gasket. However in the other contact card, the master was the engine block head (which applies the displacement control load). While the slave was the surface of the rubber gasket. Both contact cases use a slave adjustment zone tolerance of 1×10^{-5} (m). The slave adjustment zone helps prevent issues when initialization contact that was problematic for particular meshes. This tolerance zone is related to the amount that the engine block head is allowed to pass through gasket. When this was set to zero, there were problems of the engine block head not initializing contact with the rubber gasket. However with this adjustment zone tolerance, reasonable contact initialization iterations occurred.

3.3 Results

The resulting mesh can be seen in Figure 9, where the undefomred and fully closed engine head position are shown. The fillet of the gasket, becomes sharp in the deformed state. This is an indication of stress concentration. The von-mises stress is shown in Figure 10, where you can indeed see the effect of the stress concentration at these pinched corners of the gasket. The pressure of each element is seen in Figure 11, where it it seen that the maximum element pressure occur at the contact surface of the rubber gasket and engine block.


(a) Undeformed

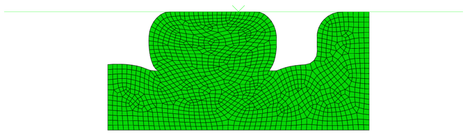

(b) Deformed

Figure 9: Results of the FE model from the unreformed to the fully closed engine head.

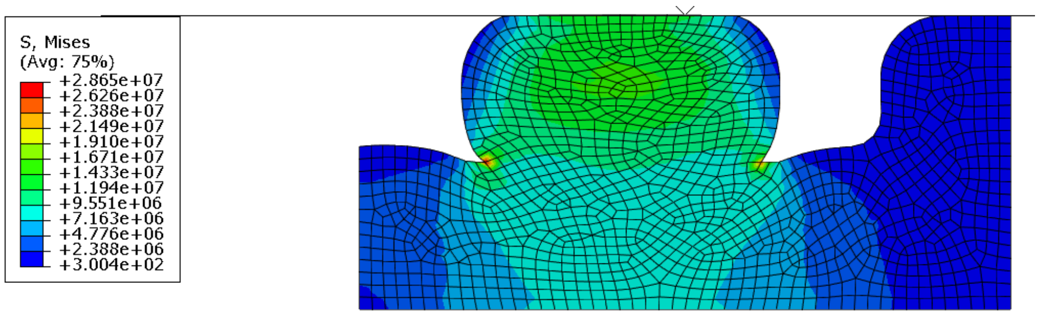


Figure 10: Von-mises stress (Pa) on the deformed rubber gasket.

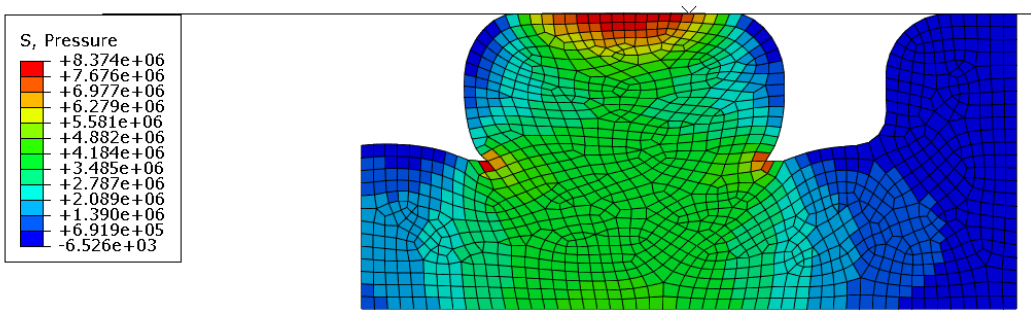


Figure 11: Pressure (Pa) of the elements on the deformed rubber gasket.

The load displacement curve of the engine block head can be seen in Figure 12. We can see that the load displacement curve is non-linear, where it appears that the response is becoming stiffer. Initially the structure is more compliant, and as the rubber deforms it takes a larger force to obtain the same displacement. This is in part geometric hardening, and the material stiffening during compression. It is unfortunate that there is no compression test data available, as the structure is primarily under compression in this example.

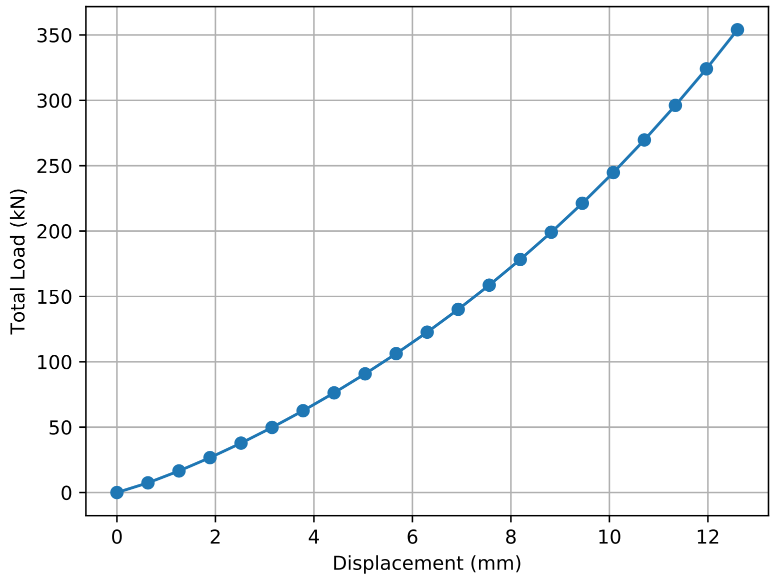


Figure 12: The load displacement curve of the engine block head.

4 Shape optimization

This section describes how shape optimization is used to modify the design of the rubber gasket, such that the gap is reduced while the contact pressure becomes as uniform as possible. The processes takes advantage of how Abaqus can related original geometry to a final mesh. This aids in the automated mesh generation, and automated post processing (which is independent of the mesh).

The overall procedure can be outlined in three parts: 1) Generate a new mesh for a given geometry, and create a new Abaqus input file. 2) Run the Abaqus input file. 3) Run the post processing routine. If any one of these three procedures fails, an objective function of infinity is passed to the optimization routine. While a successful run of all three parts will yield an objective function that is a combination of the gap closure and contact pressure.

This section goes on to describe the parametric design used to generate new geometries. The optimization formulation is then described in detail. Finally the results of the shape optimization are presented and compared with the initial design.

4.1 Parametric design

The project description allows us to alter the blue lines of Figure 1, such that the gap distance is minimized and the contact pressure is as uniform as possible. There are many possible ways to modify the geometry of the gasket, and by no means is the parametric design presented here the best possible method. However the parametric design purposed here is very simple, requiring only two design variables.

The parametric design of the Engine gasket is shown in Figure 13. There are now three design variables that can be used to describe the shape of the gasket: β_0, β_1 and R. Specifying all three of these parameters will yield a over-constrained geometry. Thus only two of the three parameter are needed to fully define the geometry, while the remaining third parameter will be determined from the other constraints. It was chosen to choose β_0 and β_1 as the two design variables, while letting R be determined by Abaqus (based on the constrained geometry). The rational for this choice is that β_0 and β_1 will be of the exact same order of magnitude, while a design radius R will have

A Python script was created which modifies an Abaqus replay file for a provided β_0 and β_1. The Abaqus replay file modifies the sketch of the Abaqus cae file, generates a new mesh, and exports a new Abaqus input file for this geometry. The Python script keeps track of whether the Abaqus replay file was successful in writing a new input file for a given geometry. The Python script used to execute and modify the Abaqus replay file is presented in Appendix C.

A couple examples of the possible meshed geometry is shown in Figure 14 by specifying β_0 and β_1. There is a substantial difference between the two examples in the geometric shape of the rubber gasket. One design using a

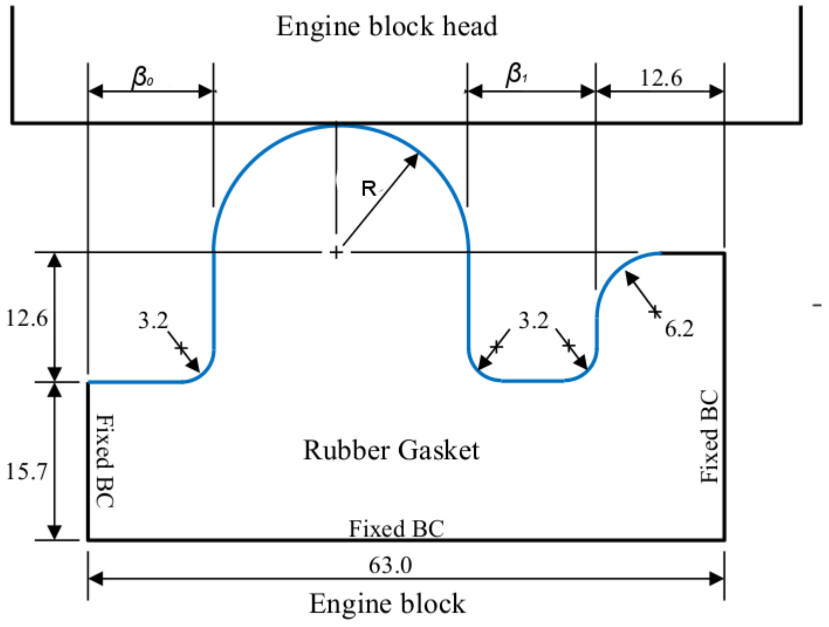
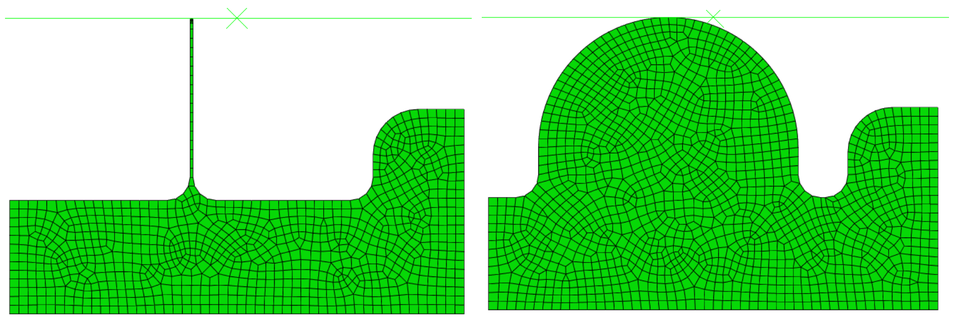


Figure 13: Parametric geometry (β_0, β_1, R) of the rubber engine gasket.

very thin (and impractical hump), while the other design uses a very large hump.

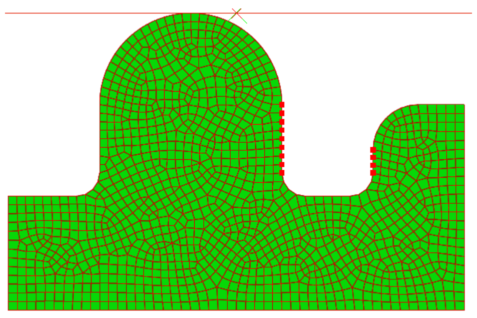


(a) $\beta_0 = 0.025$ and $\beta_1 = 0.025$ (m) (b) $\beta_0 = 0.007$ and $\beta_1 = 0.007$ (m)

Figure 14: Two examples of possible meshed geometries provided β_0 and β_1.

The last step of the parametric design is to automate the post processing based on certain node sets. The node information for contact pressure an x displacement are exported at locations of interest as shown in Figure 15.

These node locations are tied to the geometric surface, and not to any particular mesh. As the geometry and mesh changes, the surface that these nodes are located on will always be the same. This allows for the contact pressure and gap distance to be calculated for any geometry. The nodes exported for a completely different geometry and mesh are shown in Figure 16. It is interesting to compare the nodes selected between Figure 15 and Figure 16.


(a) Nodes for gap distance

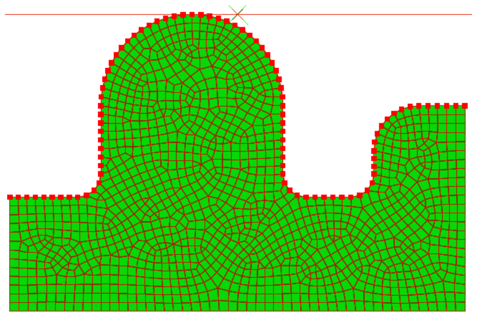
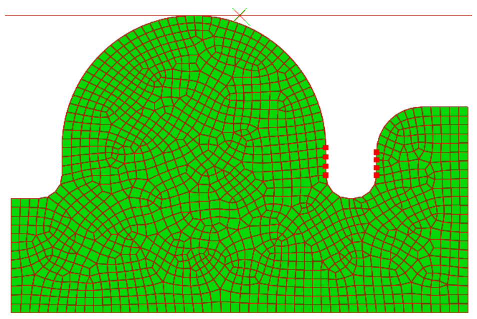

(b) Nodes for contact pressure

Figure 15: Node locations exported automatically for gap distance and contact pressure information. These nodes are selected based on the geometric surface and will change as the geometry changes.


(a) Nodes for gap distance

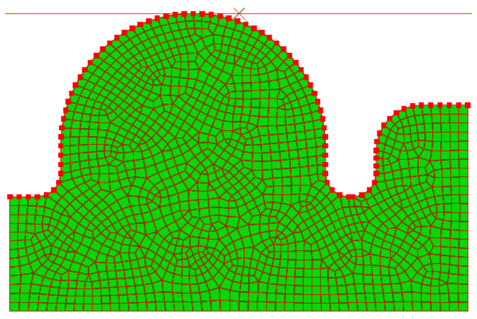

(b) Nodes for contact pressure

Figure 16: Node locations exported automatically for gap distance and contact pressure information. These nodes are selected based on the geometric surface and will change as the geometry changes.

The displacement of the gap distance nodes and contact pressure nodes (as previously illustrated) are exported automatically using a Abaqus replay file. The gap distance can be calculated from the average x displacement of the exported gap nodes. The interpretation of enforcing that the contact pressure be as uniform as possible is analogous to reducing the standard deviation of the contact pressure. Since most of the contact pressure nodes will not be in contact (i.e. will have zero contact pressure), it was decided to take the standard deviation of the nodes that have a non-zero contact pressure to represent the uniformity.

4.2 Optimization formulation

An optimization problem is formulated as

$$\text{minimize: } f(\boldsymbol{\beta}) = \left(\frac{d(\beta_0, \beta_1)}{d_0} + \frac{\sigma(\beta_0, \beta_1)}{\sigma_0} \right) \tag{14}$$

$$\text{subject to: } \textbf{Valid mesh generation and FEA run} \tag{15}$$

where d_0 is the initial gap distance, d is the gap distance as a function of β_0 and β_1, σ_0 is the initial standard deviation of the contact pressure, and σ is the standard deviation of the contact pressure for a given β_0 and β_1. The optimization goal is to find β_0 and β_1 that minimize the objective value $f(\boldsymbol{\beta})$. The initial geometry was $\beta_0 = 0.0126$ and $\beta_1 = 0.0126$, which gives that $d_0 = 0.010155$ m and $\sigma_0 = 8,341,533$ Pa.

This formulation for $f(\boldsymbol{\beta})$ will be equal to two when a geometry produces equivalent results results of the initial parameters space. The objective function will be less than two when a better design has been found. The smaller $f(\boldsymbol{\beta})$, the more improved the design is over the initial rubber gasket. It is important to note that this objective function considers the closure gap and uniform contact pressure as equal weights, and that new values are normalized from the initial geometry.

It is important that the numerical model produces a valid run, as creating new meshes will sometimes yield numerical models that are impossible to evaluate. One easy way to enforce the constraint it to modify the objective function value $f(\boldsymbol{\beta})$ based on the activity of the constraint. We can consider the constraint as a boolean value. Either the FEA produced a successful run, or it did not. The formulation of the objective function now becomes

$$\text{if FEA Succesful: } f(\boldsymbol{\beta}) = \left(\frac{d(\beta_0, \beta_1)}{d_0} + \frac{\sigma(\beta_0, \beta_1)}{\sigma_0} \right) \tag{16}$$

$$\text{else: } f(\boldsymbol{\beta}) = \infty \tag{17}$$

where if the analysis is successful than an objective value is calculated. However if any failure occurred during the analysis, an objective value of infinity will be returned. This formulation makes the design space discontinuous. For many optimization problems a discontinuous design space is problematic, however this problem is inherently discontinuous due to each design point having it's own unique mesh.

In order to solve this shape optimization problem, an optimization algorithm must be selected that can deal with discontinuous functions. There are many choices of global and local optimization algorithms available that work well on discontinuous functions such as genetic algorithms, particle swarm optimization, differential evolution, and the Nelder-Mead method. However an interesting alternative optimization algorithm based on Lipschitz functions as described here [4] was an interesting candidate. The Lipschitz function optimization algorithms can work as a global optimization method that doesn't require any hyper parameters to tune (while all the other alternative methods have hyper parameters).

The Lipschitz function optimization algorithm utilize to solve this shape optimization problem is described as the MaxLIPO+TR described online at `http://blog.dlib.net/2017/12/a-global-optimization-algorithm-worth.html`. The function is available in the dlib C++ numerical library, which has a Python interface. The authors of the dlib library were so confident in MaxLIPO+TR, that they've made it their default global optimization routine.

The shape optimization was solved using 1,000 function evlautions. The Python script to run the optimization routine is shown in Appendix D. The optimization took about four hours to complete on a AMD FX-9590 work station.

4.3 Results

The results of the shape optimization are described in Table 5. The optimization was able to find a design that reduced the gap distance d and produced a more uniform contact pressure. The objective value was reduced from 2.0 to 1.3 at the optimum. This optimum design is shown in Figure 17.

The Von-mises stress distribution on the deformed rubber gasket is shown in Figure 18. The maximum von-Mises stress was reduced form an initial 2.86×10^7 to a final of 1.95×10^7 (Pa). Additionally it can be seen that the gap distance is significantly smaller when compared to the initial designs of

	β_0 (m)	β_1(m)	d (m)	σ (Pa)	$f(\boldsymbol{\beta})$
Initial design	0.0126	0.0126	0.0102	8,341,533	2.0
Optimum design	0.0045	0.0065	0.0036	7,471,768	1.3

Table 5: Values of the initial and optimum design.

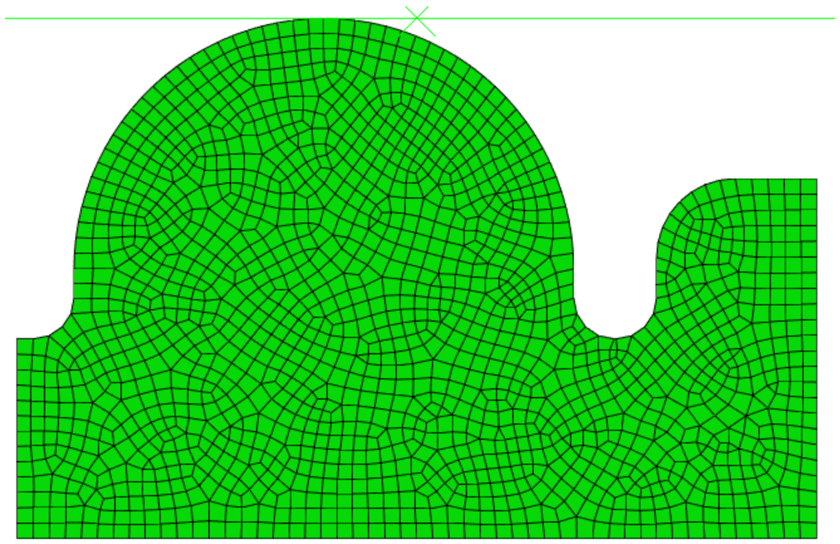


Figure 17: The optimum design of the rubber gasket according to the shape optimization.

Section 2.

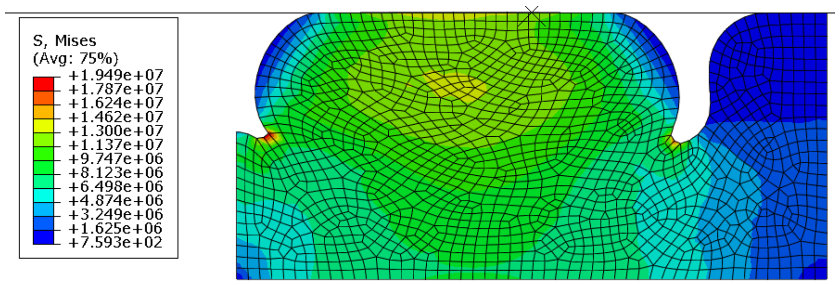


Figure 18: The von-Mieses stress of the final optimum design at the fully closed position.

The load displacement curves of the initial and optimum design are presented in Figure 19. The optimum design requires a significant amount of extra force (about 150 kN) to completely close the engine head. This extra force is primary due to the optimum design taking up a larger volume. It may be interesting to include the total load as a constraint or design objective in future work.

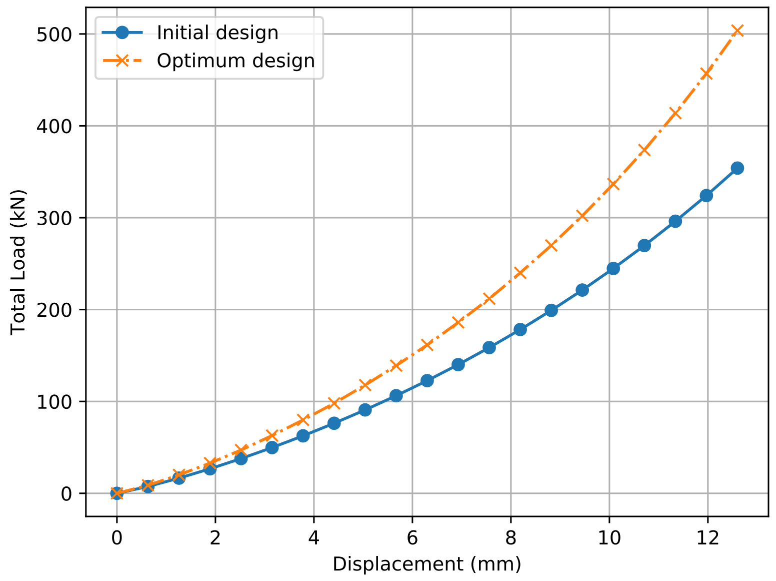


Figure 19: The load displacement curves of the engine block head for the initial and optimum design.

5 Conclusion

This report outlines how a shape optimization was performed to design a rubber gasket. The report details how material parameters and a material model were selected from test data on a rubber material. While all material models appear to be equivalent, the material chosen was a Mooney-Rivlin model determined by a curve fit in Abaqus. The rational behind this choice was related to numerical stability while having a reasonable quantitative fit value. A non-linear finite element (FE) model was constructed to replicate the physical condition of closing the engine block head onto the rubber gasket.

Shape optimization was performed to modify the shape of the rubber gasket to minimize the closure gap while making the contact pressure as uniform as possible. Optimization was performed using a Lipschitz function algorithm, because the method works well with discontinuous surfaces and has few hyper parameters. The final optimum design reduced the closure gap, and provided a more uniform contact pressure.

There are a lot of opportunities to improve upon the shape optimization routine. One could introduce more parameters to create more unique shapes. Alternatively one could explore the impact of including other important physical values. Future consideration could look at including stress, total load, and volume into the shape optimization routine.

One method to improve this application would be to provide test data from compression. The test data provided appears to be just from tension, however this application is primarily in compression. The true behavior of the material in compression is unknown, and the Mooney-Rivlin model is fit to just the tension data. The material models are extrapolated into the compression region, without truly knowing how the material behaves in this domain. New material models fit to compression data may be more accurate than the model used in this report.

A Material parameter calculation

The code to perform the least-squares Mooney-Rivlin fit, plot the strecth-stress curves, and calculate the sum-of-squares is presented below.

Python code:

```
import numpy as np
import matplotlib.pyplot as plt

strain = np.linspace(0.1,0.7,7)
tension = np.array([4.2, 6.4, 8.0, 9.2, 10.0, 11.2,
    12.6])*10**6
shear = np.array([5.0, 7.6, 9.0, 10.2, 11.2, 12.4,
    14.8])*10**6
biaxial = np.array([6.4, 9.4, 11., 13., 15., 17.6,
    21.4])*10**6

# calc lambda
```

```
l = 1.0 + strain

# calc bulk modulus k
nu = 0.495
E = tension/strain
Eavg = np.nanmean(E)
k = Eavg/ (3.0* (1.0 - (2.0*nu)))

X = np.zeros((len(strain)*3,2))
# populate regression matrix
# tension
X[0:7,0]    = 2.0*(l-(l**(-2)))
X[0:7,1]    = 2.0*(1.0-(l**(-3)))
# shear
X[7:14,0]   = 2.0*(l-(l**(-3)))
X[7:14,1]   = 2.0*(l-(l**(-3)))
# biaxial
X[14:21,0] = 2.0*(l-(l**(-5)))
X[14:21,1] = 2.0*((l**3)-(l**(-3)))


T = np.concatenate((tension,shear,biaxial))

# solve the regression matrix for A
A, resid, rank, s = np.linalg.lstsq(X,T)
T_mat = np.dot(X,A)

# abaqus results
Aabq = np.array([4576930.25,553471.564])
Tabq = np.dot(X,Aabq)
# calculate the residual
rabq = Tabq-T
# calcualte the sum of squares of residuals
eabq = np.dot(rabq.T,rabq)

# ogden N=1
Tog1 = np.array([3.0428E+06,5.4682E+06,7.4391E
    +06,9.0671E+06,1.0431E+07,
     1.1587E+07,1.2577E+07,3.9623E+06,6.9771E+06,9.3251E
         +06,1.1190E+07,1.2697E+07,
     1.3932E+07,1.4958E+07,5.6725E+06,9.6049E+06,1.2418E
         +07,1.4485E+07,1.6038E+07,
     1.7230E+07,1.8160E+07])
# calculate the residual
# calculate the residual
rog1 = Tog1-T
# calcualte the sum of squares of residuals
eog1 = np.dot(rog1.T,rog1)

# ogden N=2
Tog2 = np.array([3.0934E+06,5.5448E+06,7.5181E
    +06,9.1271E+06,1.0453E+07,
     1.1557E+07,1.2481E+07,4.0309E+06,7.0811E+06,9.4310E
         +06,1.1268E+07,1.2722E+07,
     1.3883E+07,1.4818E+07,5.7716E+06,9.7235E+06,1.2469E
         +07,1.4395E+07,1.5750E+07,
     1.6701E+07,1.7362E+07,])
# calculate the residual
# calculate the residual
rog2 = Tog2-T
# calcualte the sum of squares of residuals
eog2 = np.dot(rog2.T,rog2)

strain = np.linspace(0.0,0.7,8)
l = strain + 1.0
# Tension
plt.figure()
plt.plot(l, np.concatenate([[0.0],T[0:7]]),'ok',   label
    ='Test data')
plt.plot(l, np.concatenate([[0.0],T_mat[0:7]]), label='
    Mooney-LeastSQ')
plt.plot(l, np.concatenate([[0.0],Tabq[0:7]]), label='
    Mooney-Abaqus')
plt.plot(l, np.concatenate([[0.0],Tog1[0:7]]), label='
    Ogden N=1')
plt.plot(l, np.concatenate([[0.0],Tog2[0:7]]), label='
    Ogden␣N=2')
plt.legend()
plt.xlabel(r'Principal␣strecth␣($\lambda$)')
plt.ylabel('Nominal␣stress␣(Pa)')
plt.savefig('figs/tension.pdf')
plt.show()
# shear
plt.figure()
plt.plot(l, np.concatenate([[0.0],T[7:14]]),'ok',
    label='Test␣data')
plt.plot(l, np.concatenate([[0.0],T_mat[7:14]]), label=
    'Mooney-LeastSQ')
plt.plot(l, np.concatenate([[0.0],Tabq[7:14]]), label='
    Mooney-Abaqus')
plt.plot(l, np.concatenate([[0.0],Tog1[7:14]]), label='
    Ogden␣N=1')
plt.plot(l, np.concatenate([[0.0],Tog2[7:14]]), label='
    Ogden␣N=2')
plt.legend()
plt.xlabel(r'Principal␣strecth␣($\lambda$)')
plt.ylabel('Nominal␣stress␣(Pa)')
plt.savefig('figs/shear.pdf')
plt.show()
# biaxial
plt.figure()
plt.plot(l, np.concatenate([[0.0],T[14:21]]),'ok',
    label='Test␣data')
plt.plot(l, np.concatenate([[0.0],T_mat[14:21]]), label
    ='Mooney-LeastSQ')
plt.plot(l, np.concatenate([[0.0],Tabq[14:21]]), label=
    'Mooney-Abaqus')
plt.plot(l, np.concatenate([[0.0],Tog1[14:21]]), label=
    'Ogden␣N=1')
plt.plot(l, np.concatenate([[0.0],Tog2[14:21]]), label=
    'Ogden␣N=2')
plt.legend()
plt.xlabel(r'Principal␣strecth␣($\lambda$)')
plt.ylabel('Nominal␣stress␣(Pa)')
plt.savefig('figs/biaxial.pdf')
plt.show()
```

B Input file for coarse mesh

Abaqus input file deck for coarse mesh. This text can be coppied into a .inp file and loaded into Abaqus for reprehensibility.

```
*Heading
** Job name: Job-1 Model name: Model-1
** Generated by: Abaqus/CAE 2018
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=Part-1
*Node
      1,           0.,           0.
      2,           0., 0.0156999994
      3,  0.063000001,           0.
      4,  0.063000001, 0.0283000004
      5, 0.0565999784, 0.0283000004
      6, 0.0522159375, 0.0264840629
      7, 0.0504000001, 0.0221000221
      8, 0.0504000001, 0.0188999772
      9, 0.0472000241, 0.0156999994
     10, 0.0409999788, 0.0156999994
     11, 0.0377999991, 0.0188999772
     12, 0.0377999991, 0.0283000004
     13,  0.034109544, 0.0372095443
     14,       0.0252, 0.0408999994
     15, 0.0162904542, 0.0372095443
     16,       0.0126, 0.0283000004
     17,       0.0126, 0.0188999996
     18, 0.00939999986, 0.0156999994
     19, 0.0208485611, 0.0264722928
     20, 0.0291261263, 0.0264961831
     21, 0.0250985473, 0.0315393656
     22, 0.0520274043, 0.0181289762
     23, 0.0563221201, 0.0202204194
     24, 0.0236129202, 0.0109559335
     25, 0.0247358046, 0.0191536453
     26, 0.0416744836, 0.0109497122
     27, 0.0488618985, 0.00972666219
     28,  0.036588572, 0.00504620885
*Element, type=CPS3
 1, 24, 18,  1
 2, 22, 23,  7
 3,  7, 23,  6
 4, 27, 26, 28
 5, 23,  3,  4
 6,  6, 23,  5
 7, 22,  7,  8
 8,  4,  5, 23
 9,  1, 18,  2
10, 24,  1, 28
11, 28,  1,  3
12, 27, 28,  3
*Element, type=CPS4
13, 16, 19, 21, 15
14, 19, 16, 17, 25
15,  8,  9, 27, 22
16, 21, 20, 12, 13
17, 11, 25, 24, 10
18, 21, 13, 14, 15
19, 11, 12, 20, 25
20, 25, 20, 21, 19
21, 28, 26, 10, 24
22,  9, 10, 26, 27
23, 17, 18, 24, 25
24, 22, 27,  3, 23
*Nset, nset=d1
 11, 12
*Elset, elset=d1
 19,
*Nset, nset=d2
 7, 8
*Elset, elset=d2
 7,
*Nset, nset=fixed, generate
 1,  5,  1
*Elset, elset=fixed
  5,  8,  9, 11
*Nset, nset=Set-4, generate
  1,  28,   1
*Elset, elset=Set-4, generate
  1,  24,   1
*Nset, nset=contact_pressure
  2,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
     16, 17, 18
*Elset, elset=contact_pressure
  3,  6,  7,  8,  9, 13, 14, 15, 16, 17, 18, 19, 22, 23
*Elset, elset=_Surf-1_S3, internal
  3,  6, 16, 18
*Elset, elset=_Surf-1_S2, internal
  7,  9, 14, 18
*Elset, elset=_Surf-1_S1, internal
 15, 19, 22, 23
*Elset, elset=_Surf-1_S4, internal
 13, 17
*Surface, type=ELEMENT, name=Surf-1
_Surf-1_S3, S3
_Surf-1_S2, S2
_Surf-1_S4, S4
_Surf-1_S1, S1
** Section: Section-1
*Solid Section, elset=Set-4, material=Mooney-Rivlin
,
*End Part
**
*Part, name=Part-3
*Node
      1, -0.0250000004, 0.0408999994
      2,          -0.0137, 0.0408999994
      3, -0.00240000011, 0.0408999994
      4, 0.00889999978, 0.0408999994
      5, 0.0201999992, 0.0408999994
      6, 0.0315000005, 0.0408999994
      7, 0.0428000018, 0.0408999994
      8, 0.0540999994, 0.0408999994
      9, 0.0653999969, 0.0408999994
     10,  0.076700002, 0.0408999994
     11, 0.0879999995, 0.0408999994
*Element, type=R2D2
 1,  1,  2
 2,  2,  3
 3,  3,  4
 4,  4,  5
 5,  5,  6
 6,  6,  7
 7,  7,  8
 8,  8,  9
 9,  9, 10
10, 10, 11
*Node
     12, 0.0315000005, 0.0408999994,             0.
*Nset, nset=Part-3-RefPt_, internal
12,
*Nset, nset=top, generate
  1,  12,   1
*Elset, elset=top, generate
  1,  10,   1
*Elset, elset=Part-3, generate
  1,  10,   1
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
```

```
*Instance, name=Part-1-1, part=Part-1
*End Instance
**
*Instance, name=Part-3-1, part=Part-3
*End Instance
**
*Nset, nset=Set-2, instance=Part-3-1, generate
  1,  11,   1
*Elset, elset=Set-2, instance=Part-3-1, generate
  1,  10,   1
*Elset, elset=_CP-3-Part-1-1_S3, internal, instance=
   Part-1-1
  3,  6, 16, 18
*Elset, elset=_CP-3-Part-1-1_S2, internal, instance=
   Part-1-1
  7,  9, 14, 18
*Elset, elset=_CP-3-Part-1-1_S1, internal, instance=
   Part-1-1
 15, 19, 22, 23
*Elset, elset=_CP-3-Part-1-1_S4, internal, instance=
   Part-1-1
 13, 17
*Surface, type=ELEMENT, name=CP-3-Part-1-1
_CP-3-Part-1-1_S3, S3
_CP-3-Part-1-1_S2, S2
_CP-3-Part-1-1_S4, S4
_CP-3-Part-1-1_S1, S1
*Elset, elset=_Surf-5_SNEG, internal, instance=Part
   -3-1, generate
  1,  10,   1
*Surface, type=ELEMENT, name=Surf-5
_Surf-5_SNEG, SNEG
*Elset, elset=_s_Surf-1_S3, internal, instance=Part-1-1
  3,  6,  9, 16, 18
*Elset, elset=_s_Surf-1_S2, internal, instance=Part-1-1
  5,  7,  9, 11, 14, 18
*Elset, elset=_s_Surf-1_S1, internal, instance=Part-1-1
  8, 15, 19, 22, 23
```

```
*Elset, elset=_s_Surf-1_S4, internal, instance=Part-1-1
 13, 17
*Surface, type=ELEMENT, name=s_Surf-1
_s_Surf-1_S3, S3
_s_Surf-1_S2, S2
_s_Surf-1_S4, S4
_s_Surf-1_S1, S1
*Rigid Body, ref node=Part-3-1.Part-3-RefPt_, elset=
   Part-3-1.Part-3
*End Assembly
**
** MATERIALS
**
*Material, name=Mooney-Rivlin
*Hyperelastic, mooney-rivlin
 4.57693e+06,      553472., 1.95568e-09
**
** INTERACTION PROPERTIES
**
*Surface Interaction, name=IntProp-1
1.,
*Friction, slip tolerance=0.005
 0.05,
*Surface Interaction, name=IntProp-2
1.,
*Friction, slip tolerance=0.005
 0.05,
**
** CONTACT INITIALIZATION DATA
**
*Contact Initialization Data, name=CInit-1
**
** BOUNDARY CONDITIONS
**
** Name: BC-1 Type: Displacement/Rotation
*Boundary
Part-1-1.fixed, 1, 1
Part-1-1.fixed, 2, 2
```

```
**
** INTERACTIONS
**
** Interaction: Int-2
*Contact Pair, interaction=IntProp-1, type=SURFACE TO
  SURFACE, adjust=1e-05
s_Surf-1, Surf-5
** Interaction: self_contact
*Contact Pair, interaction=IntProp-1, type=SURFACE TO
  SURFACE, adjust=1e-05
s_Surf-1, s_Surf-1
**
  ----------------------------------------------------------------------


**
** STEP: Step-1
**
*Step, name=Step-1, nlgeom=YES, inc=2000
*Static, direct
0.05, 1.,
**
** BOUNDARY CONDITIONS
**
** Name: BC-2 Type: Displacement/Rotation
*Boundary
Set-2, 1, 1
Set-2, 2, 2, -0.0126
**
** OUTPUT REQUESTS
**
*Restart, write, frequency=0
**
** FIELD OUTPUT: F-Output-1
**
*Output, field
*Node Output
CF, RF, U
*Element Output, directions=YES
```

```
HP, LE, MISES, P, PE, PEEQ, PEMAG, S
*Contact Output
CDISP, CSTRESS
**
** FIELD OUTPUT: F-Output-2
**
*Element Output, directions=YES
HP, MISES, P
*Contact Output
CSTRESS,
**
** HISTORY OUTPUT: H-Output-1
**
*Output, history, variable=PRESELECT
*End Step
```

C Abaqus write input for given geometry

The Python script used to modify the Abaqus replay file template, and then run the replay file, for a specified β_0 and β_1:

```
from __future__ import print_function
import os

def new_mesh_run_model(x):
    # set the state of the jobs
    success = False
    # modify the pre process template file
    with open('write_inp_template.py', 'r') as d, open(
       'write_inp.py', 'w') as f:
        # load the template
        data = d.read()
        # replace the two beta values with the x value
        data = data.replace('BETA1VALUE',str(x[0]))
        data = data.replace('BETA2VALUE',str(x[1]))
        # write the input file
        f.write(data)
    # run the prepprocess file
    val = os.system('abaqus cae noGUI=write_inp.py')
    print(val)
    if val ==0:
        # the mesh was a success
        # execute the job
        val = os.system('abq2018 job=Job-1 interactive 
           cpus=1 ask_delete=OFF > /dev/null')
        # interactive means abaqus waits for the job to
            finish
        # > /dev/null sends the print out to a non
           existant drive in order to
        # keep a clean log file
        print(val)

        # read the last line of the job stats fileName
        with open('Job-1.sta','r') as f:
            read_data = f.readlines()
            print(read_data[-1])
        if read_data[-1] == ' THE ANALYSIS HAS 
           COMPLETED SUCCESSFULLY\n':
            # the analysis was completed successfully
            success = True

    if success== True:
        # execute post process
        return True
    else:
        return False

new_mesh_run_model([0.007, 0.007])
```

The Abaqus replay template file which is modeifeid by the previous Python script:

```
# -*- coding: mbcs -*-
#
# Abaqus/CAE Release 2018 replay file
# Internal Version: 2017_11_07-12.21.41 127140
# Run by cj on Wed Mar 21 17:01:12 2018
#

# from driverUtils import executeOnCaeGraphicsStartup
# executeOnCaeGraphicsStartup()
#: Executing "onCaeGraphicsStartup()" in the site
    directory ...
from abaqus import *
from abaqusConstants import *
mydir = '/home/cj/Dropbox/phdWork/courseWork/2018Spring
    /EGM6352AdvanceFEA/HW/Project1/abaqus_work/opt_demo/
    '
session.Viewport(name='Viewport: 1', origin=(0.0, 0.0),
     width=273.84375,
     height=207.680557250977)
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
from caeModules import *
from driverUtils import executeOnCaeStartup
executeOnCaeStartup()
session.viewports['Viewport: 1'].partDisplay.
    geometryOptions.setValues(
     referenceRepresentation=ON)
openMdb(
     pathName=mydir+'opt_struct.cae')
#: The model database "/home/cj/Dropbox/phdWork/
    courseWork/2018Spring/EGM6352AdvanceFEA/HW/Project1/
    abaqus_work/opt_struct.cae" has been opened.
session.viewports['Viewport: 1'].setValues(
    displayedObject=None)
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(
    displayedObject=p)
p = mdb.models['Model-1'].parts['Part-1']
s = p.features['Shell planar-1'].sketch
mdb.models['Model-1'].ConstrainedSketch(name='__edit__'
    , objectToCopy=s)
s1 = mdb.models['Model-1'].sketches['__edit__']
```

```
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions,
    s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p.projectReferencesOntoSketch(sketch=s1,
     upToFeature=p.features['Shell planar-1'], filter=
        COPLANAR_EDGES)
d[6].setValues(value=BETA1VALUE, )
d[12].setValues(value=BETA2VALUE, )
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
p.features['Shell planar-1'].setValues(sketch=s1)
del mdb.models['Model-1'].sketches['__edit__']
p = mdb.models['Model-1'].parts['Part-1']
p.regenerate()
#: Warning: Failed to attach mesh to part geometry.
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(
    displayedObject=a)
session.viewports['Viewport: 1'].assemblyDisplay.
    setValues(
      optimizationTasks=OFF, geometricRestrictions=OFF,
        stopConditions=OFF)
session.viewports['Viewport: 1'].partDisplay.setValues(
    mesh=ON)
session.viewports['Viewport: 1'].partDisplay.
    meshOptions.setValues(
      meshTechnique=ON)
session.viewports['Viewport: 1'].partDisplay.
    geometryOptions.setValues(
      referenceRepresentation=OFF)
p1 = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(
    displayedObject=p1)
p = mdb.models['Model-1'].parts['Part-1']
p.generateMesh()
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(
    displayedObject=a)
```

```
mdb.jobs['Job-1'].writeInput(consistencyChecking=OFF)
#: The job input file has been written to "Job-1.inp".
```

D Shape optimization script

The Python script to perform the shape optimization:

```
from __future__ import print_function
import numpy as np
import pandas as pd
import os
import dlib

def new_mesh_run_model(x):
    # set the state of the jobs
    success = False
    # modify the pre process template file
    with open('write_inp_template.py', 'r') as d, open(
       'write_inp.py', 'w') as f:
        # load the template
        data = d.read()
        # replace the two beta values with the x value
        data = data.replace('BETA1VALUE',str(x[0]))
        data = data.replace('BETA2VALUE',str(x[1]))
        # write the input file
        f.write(data)
    # run the prepprocess file
    val = os.system('abaqus cae noGUI=write_inp.py')
    print(val)
    if val ==0:
        # the mesh was a success
        # execute the job
        val = os.system('abq2018 job=Job-1 interactive 
           cpus=4 ask_delete=OFF > /dev/null')
        # interactive means abaqus waits for the job to
            finish
        # > /dev/null sends the print out to a non
            existant drive in order to
        # keep a clean log file
        print(val)

        # read the last line of the job stats fileName
        try:
            with open('Job-1.sta','r') as f:
                read_data = f.readlines()
                print(read_data[-1])
            if read_data[-1] == ' THE ANALYSIS HAS 
                COMPLETED SUCCESSFULLY\n':
                # the analysis was completed
                    successfully
                success = True
        except:
            success = False
    if success== True:
        # execute post process
        return True
    else:
        return False

def post_process(x):
    # run the post_process script
    val = os.system('abaqus cae noGUI=write_res.py')
    print(val)
    # if the script was okay
    if val == 0:
        # calc the average node location for nodes on
            d1
        d1 = pd.read_csv('d1.csv')
        d1u = np.array(d1.values[:,-1])
        d1x = np.array(d1.values[:,-7])
        d1res = np.mean(d1x+d1u)
        # calc the average node location for nodes on
            d2
        d2 = pd.read_csv('d2.csv')
        d2u = np.array(d2.values[:,-1])
        d2x = np.array(d2.values[:,-7])
        d2res = np.mean(d2x+d2u)
        # d is the difference between d1 and d2
        d = d2res - d1res # from initial =
            0.010154547785833336

        # compute the standard deviation of the contact
             pressure
        contact_pressure = pd.read_csv('
            contact_pressure.csv')
        p = contact_pressure.values[:,-1]
        # ignore zero contact pressure surface elements
        p[p==0.0]=np.nan
        p = p.astype(np.float)
        pstd = np.nanstd(p) # from initial =
            5188183.430865008
        print('var:', x)
        print('obj:',d,pstd)
        return (d / 0.010154547785833336) + (pstd
            /8341533.54401)
    else:
        # if the job wasn't succesful return inf
        return np.inf

def my_obj_fun(x0,x1):
    x = [x0,x1]
    # run the pre processing script
    success = new_mesh_run_model(x)
    if success == True:
        # run the post processing script
        obj = post_process(x)
    else:
        obj = np.inf
    return obj

#val1 = my_obj_fun(0.025, 0.025)
#val2 = my_obj_fun(0.0126, 0.0126)
```

```
#val3 = my_obj_fun(0.013, 0.0008)

# run optimization using Lipschitz functions
x,y = dlib.find_min_global(my_obj_fun
    ,[0.003,0.003],[0.035,0.035],1000)

print('******** OPT FOUND *******')
print('X: ', x)
print('Y: ', y)
```

References

[1] M. C. Boyce and E. M. Arruda, "Constitutive Models of Rubber Elasticity: A Review," *Rubber Chemistry and Technology*, vol. 73, no. 3, pp. 504–523, 2000. [Online]. Available: https://doi.org/10.5254/1.3547602

[2] P. A. L. S. Martins, R. M. Natal Jorge, and A. J. M. Ferreira, "A Comparative Study of Several Material Models for Prediction of Hyperelastic Properties: Application to Silicone-Rubber and Soft Tissues," *Strain*, vol. 42, no. 3, pp. 135–147, 2006. [Online]. Available: http://dx.doi.org/10.1111/j.1475-1305.2006.00257.x

[3] N.-H. Kim, *Introduction to nonlinear finite element analysis.* Springer Science & Business Media, 2014.

[4] C. Malherbe and N. Vayatis, "Global optimization of Lipschitz functions," *arXiv preprint arXiv:1703.02628*, 2017.