Determination of optimal support locations to maximize fundamental frequency of printed circuit boards*

Samuel Avila, Greg Vernon, & Chris Whetten

Abstract

Printed circuit boards (PCB) are critical components in many aerospace applications. In these applications harsh dynamic environments, such as shock and vibration, are encountered and it is critical that the electronic assemblies endure these conditions. An effective method for minimizing the detrimental impact of these conditions on a PCB is to increase the lowest natural frequency of the PCB through the use of structural supports. In this report we describe a global optimization method, based on surrogate models, that was found to be effective at finding optimal support locations on a PCB. Using a global optimization method within the DAKOTA optimization solver, wrapping a smooth-spline based finite element software, proves effective at handling unreliable gradients and multimodal solution space.

1 Purpose

A primary purpose of this project is to apply the open-source DAKOTA optimization framework to the smooth-spline finite element software being developed by Coreform. DAKOTA is the primary *Verification, Validation, and Uncertainty Quantification (VV & UQ)* platform used and developed by Sandia National Labs. Coreform currently services several U.S. Department of Energy customers that have requested to use the DAKOTA optimization framework with Coreform's tools. This effort will help identify required features in Coreform's software to support the use of DAKOTA as an optimization platform.

Due to current limitations of Coreform's software capabilities, a simple exemplar problem is identified that will allow for various job submission categories, as well as DAKOTA's optimization/space-search algorithms to be evaluated. This project can then be used as a tutorial to train future users of Coreform's finite element code in the use of DAKOTA for automated job submissions, VV & UQ efforts, and optimization.

2 Exemplar Problem: Optimal placement of printed circuit board supports

2.1 Motivation

Printed circuit boards are the principle component of nearly every modern electronic assembly. PCBs not only provide a platform on which to mount electronic the various electronic components of an assembly but also act as a substrate through which to electronically connect the components. PCBs are a planar geometry, having a thickness that is usually much smaller than its other two dimensions[1]. In aerospace applications PCB assemblies are subject to high dynamic load environments that can induce fatigue-based and/or strength-based failure modes in PCBs. The predominant strategy to decrease the effects of the dynamic loads on a PCB is to increase the first natural frequency (i.e. the *fundamental frequency*) as much as possible. This is primarily done through the use of various *support mechanisms* such as screw mounts or wedge supports. An example of a PCB with holes to accommodate screw mounts is shown in figure 1.

2.2 Prior Work

Won and Park[1] presented an method for identifying optimal, variable finite-stiffness support positions, on a plate. Their method calculates the sensitivity of the eigenvalue at each configuration and then produces a new configuration based on this sensitivity. Their

*This report was prepared to partially fulfill the course requirements for ME575: Optimization Techniques at Brigham Young University in Provo, Utah. ***This report has not undergone peer-review.***

[1]For instance a Micro-Star 845E Max mainboard has dimensions 300mm x 200mm x 1.5mm

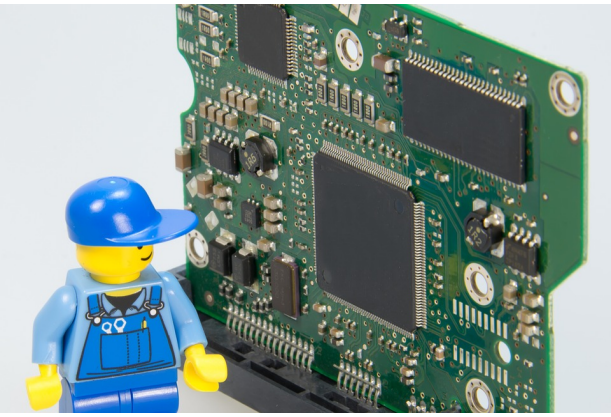


Figure 1: Example of PCB with support locations located near the board's boundary.
https://callnerds.com//motherboard-repair-replacement

method uses a finite element analysis, with a static mesh, to calculate the eigenvalues at each configuration.

Ong and Lim [2] presented an approach for locating the optimal placement of infinite stiffness supports on beams and plates. Their approach places point constraints at positions that eliminates lower modes from the original configuration. These points are chosen by introducing the constraints along the nodal lines of the modeshape corresponding to the first eigenfrequency occurring at or above the desired new fundamental frequency. Then, for 2D cases, the *average driving point residues* calculated from higher modes are used to determine the optimal locations along the nodal lines. The ANSYS finite element analysis code is used for computing the natural frequencies, mode shapes, and average driving point residues.

Ou and Mak [3] describe a global optimization approach for determining the configuration of *external* boundary conditions for a plate that provide optimal values *across multiple frequencies*. Their methodology utilizes a genetic algorithm that includes a finite element analysis to evaluate the objective function.

Chen [4] describes a method for determining optimal placement of support locations for a PCB loaded with heavy components - such as a cooling fan. Chen's method constructs unique feasible domains for the support locations and uses a commercial engineering software (ANSYS Workbench) to control the optimization algorithm (ANSYS optimization module) and objective function evaluation (ANSYS FEA). Chen converts the constrained problem to an unconstrained problem via penalty functions and uses the nonlinear conjugate gradient method.

3 Implementation

3.1 General Eigenvalue Problem

The fundamental frequency is found by solving the generalized eigenvalue problem (GEP) [5] posed in equation 1, which is sometimes written in the form shown in equation 2.[2]

$$(\mathbf{A} - \lambda\mathbf{B})\,\mathbf{x} = \mathbf{0} \tag{1}$$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \tag{2}$$

In the general case, $\mathbf{A}$ and $\mathbf{B}$ are infinite-dimensional linear operators. However, in the finite element method, the PDE's domain is discretized into cells of finite size (i.e. finite elements) which results in a discrete solution space - thus the linear system is comprised of finite-dimensional operators which are represented as matrices. The discrete problem is then written in equation 3. For the discrete problem there are $\dim(\mathbf{A})$ total solutions, with $\operatorname{rank}(\mathbf{A})$ unique solutions. Indexing the total solutions in order from the lowest eigenvalue to the largest the equation can be rewritten 3 with index notation in equation 4.

$$\left(\mathbf{A}^{\mathrm{h}} - \lambda^{\mathrm{h}}\mathbf{B}^{\mathrm{h}}\right)\mathbf{x}^{\mathrm{h}} = \mathbf{0}^{\mathrm{h}} \tag{3}$$

$$\left(\mathbf{A}^{\mathrm{h}} - \lambda_i^{\mathrm{h}}\mathbf{B}^{\mathrm{h}}\right)\mathbf{x}_i^{\mathrm{h}} = \mathbf{0}^{\mathrm{h}} \tag{4}$$

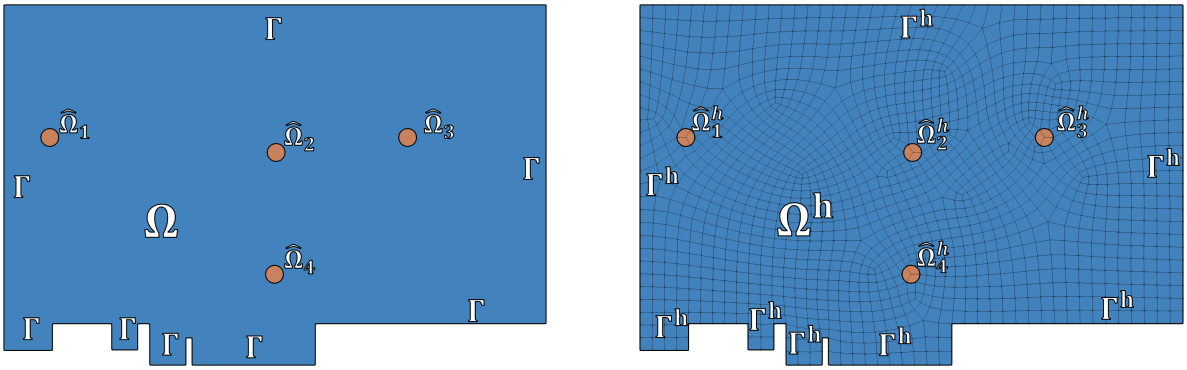
Finally, for the PDEs that arise in structural dynamics applications $\mathbf{A}$ is called the *stiffness matrix* and is often written as $\mathbf{K}$; $\mathbf{B}$ is called the *mass matrix* and is written as $\mathbf{M}$; the entries of the eigenvectors represent displacements and $\mathbf{x}$ is thusly written as $\mathbf{u}$. This final form of the general eigenvalue problem is provided in equation 5. Once the generalized eigenvalue problem has been solved for λ_i^{h} the angular frequency of the system ω_i can be determined through the relation in equation 6. This process for constructing and solving the general eigenvalue problem, as well as computing the objective function for the optimization routine, is shown in figure 3.

$$\left(\mathbf{K}^{\mathrm{h}} - \lambda_i^{\mathrm{h}}\mathbf{M}^{\mathrm{h}}\right)\mathbf{u}_i^{\mathrm{h}} = \mathbf{0}^{\mathrm{h}} \tag{5}$$

$$\lambda_i = \omega_i^2 \tag{6}$$

For the posed problem, there are no large-displacement effects (e.g. no initial stresses) and so $\mathbf{K}$ can be assumed to be symmetric positive definite, although in some instances, depending on the quality

[2] When $\mathbf{B} \equiv \mathbf{I}$ equation 2 simplifies to what is perhaps the more commonly recognized eigenvalue problem: $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$.



(a) Continuous (b) Discrete

Figure 2: Continuous and discrete problem definitions of the optimization result for a four-support configuration

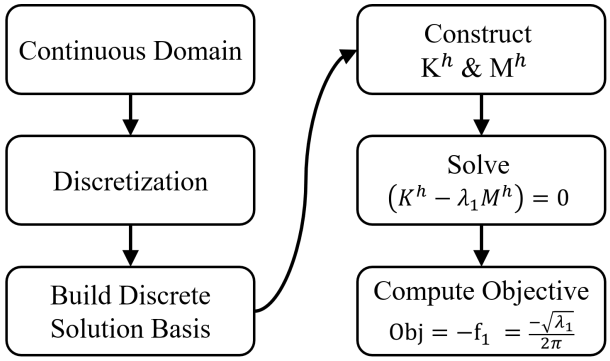


Figure 3: High-level description of solving the generalized eigenvalue problem within the optimization process.

of the finite elements and the boundary conditions, may only be semidefinite.[3]

To reduce on computational costs an efficient method is to extract only the lowest (i.e. fundamental) eigenvalue of the problem. The most obvious method would be to use solvers available in the finite element solver, however Coreform Crunch does not yet support eigenvalue problems therefore we were limited to writing the linear system to disk and solved using a 3rd-party software. To this end, solvers were evaluated for the GEP in Python, MATLAB, and Julia and these results are provided in table 1.

The results showed little variance in the accuracy of the fundamental frequency, but also demonstrated that MATLAB is the most performant. Further investigation found that the techniques available in Julia scaled poorly as the size of the linear system increased, whereas MATLAB's performance scaled well - thus MATLAB was chosen as the eigensolver.

Language	**Relative Error**	**Time (s)**
Python	9.2659e-11	30
MATLAB	0.	0.15*
Julia (eigen)	1.0078e-10	12†
Julia (LOBPCG)	6.0682e-11	6.8†

Table 1: Comparing languages for solving the generalized eigenvalue problem. (*) Does not include startup time. (†) Includes JIT compilation time

3.2 Nonlinear Inequality Constraints

Nonlinear inequality constraints were defined to restrict the support placement to within the feasible region, Ω_F, of the PCB (or disk). Each support is assigned a single nonlinear inequality constraint, which is computed in Coreform Flex by first querying whether the center of the support lies within the CAD BREP surface associated with the feasible region. The minimum distance between each support and each boundary is calculated and its additive inverse is used if the support is on the surface - to return a satisfied constraint. Thus this constraint returns a value whose magnitude is the distance from the nearest boundary of the feasible region and whose sign indicates whether the support is within the feasible region (negative – satisfied constraint) or is outside the feasible region (positive – violated constraint).

Listing 1: Python function which computes the nonlinear inequality constraint described in figure 5

[3] A matrix is positive definite **if and only if** all its eigenvalues are positive. A matrix is semi-definite **if and only if** all its eigenvalues are *non-negative* – the finite element solution includes rigid-body modes occurring at $\lambda = 0$.

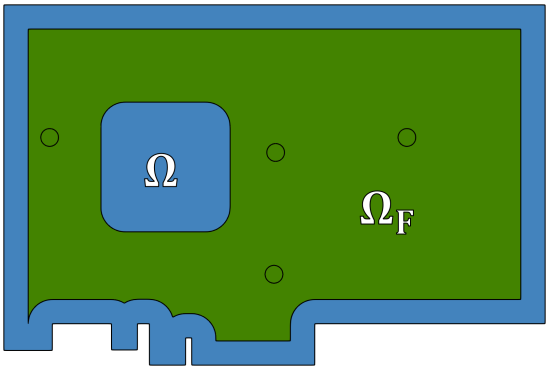


Figure 4: Geometric domain (Ω – blue) representing realistic PCB with a feasible domain (Ω_F – green) that represents a minimum distance that the center of a support may be placed from the board's edge and from a critical board-mounted component. This requirement is enforced via nonlinear inequality constraints. The four circles represent the support locations as determined from the optimization process.

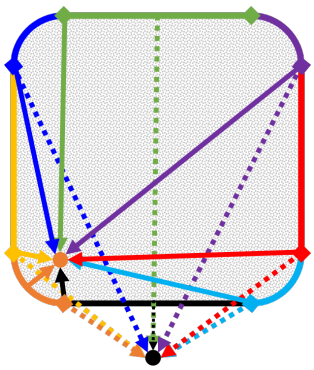


Figure 5: Graphical depiction of calculating the "support on surface" nonlinear inequality constraint. In this example the supports (circles) are colored by the boundary curve to which they are closest. The orange support lies on the surface, satisfying the constraint, while the black support lies outside the surface, violating the constraint.

```
def computeNonlinearConstraint(x,y):
    nlcon =numpy.zeros(len(x))
    target_surface =cubit.surface(2) # Feasible domain
         surface is id = 2
    vertex_on_surface =[False for i in range(0,len(x))]
    # First, determine whether the nonlinear constraint is
        satisfied
    for i in range(0,len(x)):
        vertex_on_surface[i] =target_surface.
            point_containment([x[i], y[i], 0.])
    # Second, determine the magnitude of the nonlinear
        constraint value
    # which is the distance of the point to the closest
        curve
    cid =cubit.parse_cubit_list("curve", "in vol 2")
    for i in range(0,len(x)):
        dist =numpy.zeros(len(cid))
        pXYZ =numpy.array([x[i], y[i], 0.])
        for c in range(0,len(cid)):
            C =cubit.curve(cid[c])
            cpXYZ =numpy.array(C.closest_point(pXYZ))
            dist[c] =numpy.linalg.norm(cpXYZ -pXYZ)
        if vertex_on_surface[i] ==True:
            # Nonlinear constraint is satisfied
            nlcon[i] =-1. *numpy.min(dist)
        else:
            # Nonlinear constraint not satisfied
            nlcon[i] =+1. *numpy.min(dist)
    return nlcon
```

3.3 Summary

The optimization problem is summarized in equation 7. In terms of the physical definition (x_i, x_{i+1}) corresponds to the (x, y) position of a support, of which there are N_s. The domain of the PCB is denoted Ω, while the region of the PCB within which a support's center must be contained (i.e. feasible domain) is denoted Ω_F. The *bounding box* of Ω is used as the lower and upper bounds, and is denoted as $\min_{\mathbf{x}} \Omega$ and $\max_{\mathbf{x}} \Omega$ for the minimum and maximum extents of the bounding box.

$$
\begin{aligned}
&\underset{x \in \Omega_F \subset \Omega}{\arg\min} && \frac{-\sqrt{\lambda_1(x)}}{2\pi} \\
&\text{subject to} && (x_i, x_{i+1}) \in \Omega_F, && i = 1, 3, \ldots, 2\mathrm{N_s} - 1, \\
& && x_i \geq \min_{\mathbf{x}} \Omega, && i = 1, 2, \ldots, 2\mathrm{N_s}, \\
& && x_i \leq \max_{\mathbf{x}} \Omega, && i = 1, 2, \ldots, 2\mathrm{N_s}
\end{aligned}
\tag{7}
$$

4 Optimization Workflow

4.1 DAKOTA

While DAKOTA has some abilities to directly interface with some 3rd-party software (such as MATLAB, Python, and Sandia's *SIERRA* finite element package), this functionality doesn't exist for Coreform's Flex and Crunch software. Thus, after testing optimization capabilities using DAKOTA on a simple surrogate model (see section 4.4), DAKOTA's *fork simulation interface* was investigated. This approach uses the operating system to create new processes that communicate with DAKOTA through parameter and response files, initiating the simulation software via its own standard invocation procedure (as a "black box") which can then be coordinated with a variety of tools for pre- and post-processing. Transfer of variables and response data between DAKOTA and the simulator code then occurs through reading and writ-

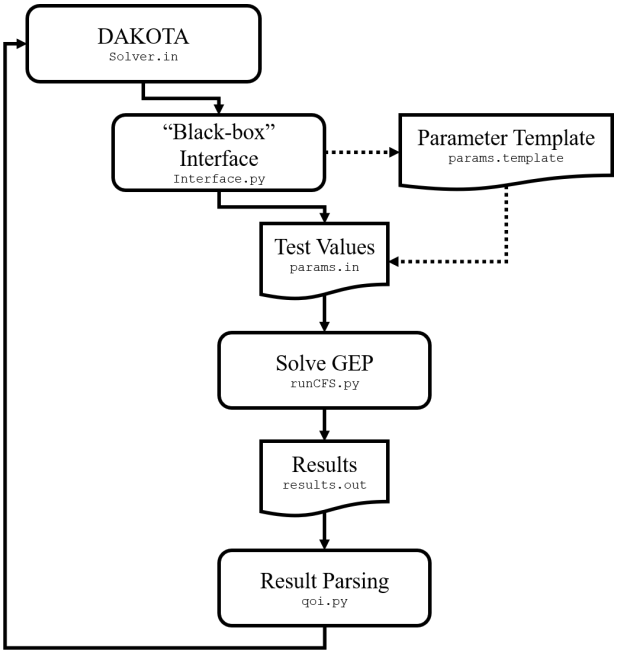


Figure 6: Process-flow diagram of DAKOTA's "Black-Box" interface, as implemented in the project.

ing of text or standard data files [6]. Figure 6 shows the process flow-diagram that occurs when DAKOTA is executed. As DAKOTA input files, and the ancillary black-box functions, can become extremely verbose when including multiple independent variables and constraints, templated versions of these files were developed which are then used to automatically generate a unique set of files for a given problem definition, see listings 2, 3, and 4 for an example of this process.

A significant challenge encountered was that, since Coreform Crunch currently only supports strongly-enforced boundary conditions, each function evaluation requires a unique discretization leading to linear systems that are not equivalent, even prior to the application of the Dirichlet boundary conditions. Numerical finite differencing was thus highly unreliable and often would result in DAKOTA errantly "finding" local minima at seemingly every trial point.

The objective function evaluation is expensive, requiring between 10s - 180s depending the problem size, we investigated[4] so we sought solution methods that would minimize the amount of sequential function evaluations needed to converge to a solution. Additionally, whenever a solution method was compatible with DAKOTA's asynchronous evaluation capabilities, we used this functionality to execute multiple objective function evaluations simultaneously. On our workstation we were able to concurrently evaluate 20 evaluations reliably.

We determined DAKOTA's `efficient_global` method to be the most efficient method, while also able to robustly find good solutions. The efficient global optimization (EGO) method in DAKOTA utilizes a surrogate modeling approach, using a Gaussian process approximation of the objective function within each of its sequential iterations. EGO constructs an *expected improvement function* (EIF), which provides insight into how much a given new trial might improve the objective function. EGO then uses DAKOTA's *DIvision of RECTangles* (DIRECT) method to perform the optimization of the EIF function. Since the EIF provides information regarding potential trials that minimize the objective function, as well as regions of the design space with high uncertainty, it provides a good balance between between exploring and exploiting. Finally, EGO handles constraints via an augmented Lagrangian merit function.

Listing 2: Snippet of templated Python script – `interface_template.py`

```
...
mdl_in_path ="params.template"
mdl_out_option =mdl_out_option_output_file

#{define_qois}

cmd_line ="Python3 runCFS.py ${input_file} ${output_file}"
cmd_in_repl ="${input_file}"
...
```

Listing 3: Function `makeInput.py` that parses listing 2 for `#{define_qois}` and replaces with problem-specific code

```
def define_qois(fLines, num_pins):
    qoi_str ='qois =[qoi.QoiAnchor("objective", 1, qoi.
          FIELDS, 1, qoi.FIELDS, qoi.AFTER, "ObjVal"),' +"\
          n"
    for i in range(0,num_pins):
        qoi_str +=" "*8 +'qoi.QoiAnchor("nlcon_' + str(i
              +1) + '", 1, qoi.FIELDS, 1, qoi.FIELDS, qoi.
              AFTER, "nlcon_' + str(i+1) + '"),' +"\n"
    qoi_str +=" "*8 +"]"

    for i in range(0,len(fLines)):
        fLines[i] =fLines[i].replace("#{define_qois}",
              qoi_str)
    return fLines
```

Listing 4: Auto-generated script, `interface.py`, created via listing 3 operating on listing 2

```
mdl_in_path ="params.template"
```

[4]Problem sizes ranged from ~60 $\mathcal{P}^2\mathcal{C}^1$ finite elements to ~1500 $\mathcal{P}^4\mathcal{C}^3$ finite elements.

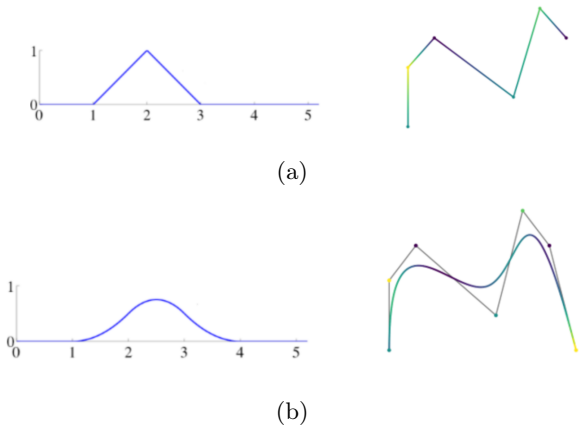


Figure 7: Comparison of linear shape functions and associated geometry (a) with a smooth polynomial spline function (b).

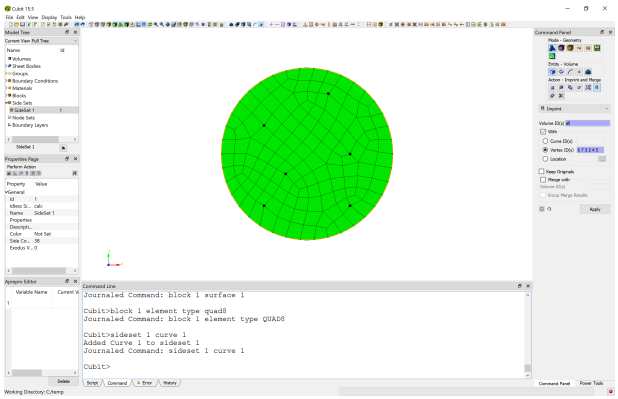


Figure 8: Coreform Flex is based on *Trelis*, which is the commercial variant of the *Cubit* meshing software developed by Sandia National Laboratories

```
mdl_out_option =mdl_out_option_output_file

qois =[qoi.QoiAnchor("objective", 1, qoi.FIELDS, 1, qoi.
     FIELDS, qoi.AFTER, "ObjVal"),
       qoi.QoiAnchor("nlcon_1", 1, qoi.FIELDS, 1, qoi.
            FIELDS, qoi.AFTER, "nlcon_1"),
       qoi.QoiAnchor("nlcon_2", 1, qoi.FIELDS, 1, qoi.
            FIELDS, qoi.AFTER, "nlcon_2"),
       qoi.QoiAnchor("nlcon_3", 1, qoi.FIELDS, 1, qoi.
            FIELDS, qoi.AFTER, "nlcon_3"),
       qoi.QoiAnchor("nlcon_4", 1, qoi.FIELDS, 1, qoi.
            FIELDS, qoi.AFTER, "nlcon_4"),
       ]

cmd_line ="Python3 runCFS.py ${input_file} ${output_file}"
cmd_in_repl ="${input_file}"
```

4.2 Coreform Flex

Coreform is primarily focused on developing a suite of tools that support a spline-based finite element analysis workflow. Traditional finite element analysis primarily uses linear or C^0 Lagrange formulations as the underlying shape functions for their discretization. A fundamental characteristic of all these basis functions is that they have discontinuous derivatives (C^0) on element boundaries. An alternative method introduced by Hughes et. al. [7] uses higher-order smooth-polynomials as the basis functions for analysis – a method commonly referred to as *isogeometric analysis* (IGA). A simple comparison of traditional linear elements with smooth splines is shown in Figure 7.

Recent developments in the field of splines allow for higher-order smooth-spline functions defined over unstructured and non-conforming meshes (T-splines or U-splines) [8]. When performing dynamic analysis however, higher order functions are rarely used since they often exhibit poor spectral behavior which negatively affects dynamic simulation. Recently it has been shown that U-splines with mixed continuity can alleviate the poor spectral properties shared by both higher order Lagrange elements and traditional smooth splines [9]. These higher order U-splines were used as the basis for this dynamic frequency analysis though no explicit comparison of this basis with traditional methods is included.

Coreform Flex is used to produce both the topological layout (i.e. mesh) and the underlying basis functions needed for analysis. In this current work we apply the Dirichlet boundary conditions using strong enforcement which requires special consideration when building the mesh topology and constructing the spline basis. IGA, just like traditional FEA, requires interpolatory functions (i.e. nodes) to exist at Dirichlet boundaries in order to be enforced strongly. Using the meshing capabilities in Coreform Flex that were developed by Sandia National Laboratories for their Cubit software, the locations of mesh vertices can be enforced to conform to the support locations and then using standard paving algorithms to automatically and robustly generate the mesh. Then, using the mixed continuity basis construction available for U-splines, the edges adjacent can be creased to the vertex of interest to C^0 while leaving other element boundaries (where possible) at higher (C^1 or greater) continuity. It should be noted that in order to construct a well-behaved basis, additional creasing is often necessary to obtain other requirements such as local linear independence and partition of unity. Several modifications were made in order to achieve the robustness required by this optimization workflow.

4.3 Coreform Crunch

Once the mesh and the basis have been built, the code automatically constructs the input necessary for Coreform's solver: Coreform Crunch. This is relatively straightforward since there are no applied traction forces or time integration. A simple linear elastic material model and a solid element formulation are used. In general it would be possible to use standard shell formulations, however, constructing the mass matrix needed by equation 5 would require an additional integration through the thickness which would require additional work. It is instead easier to extrude the two-dimensional mesh created in Flex and create a tensor product basis with a one dimensional spline. Like many other parts of this workflow, since no explicit integration exists the files are used as to communicate between the various Coreform components. The standard file format used by both Flex and Crunch are JSON files. This workflow uses a Python script that reads in a JSON file with basic mesh information from Flex and then automatically parses and writes out a new JSON file for use in Crunch.

The primary role of the finite element solver in our workflow is to assemble $\mathbf{K}^{\mathrm{h}}$ and $\mathbf{M}^{\mathrm{h}}$. Coreform Crunch uses C++ as its primary language and currently uses the PETSc linear algebra package available for C++. As of this report there is no way to solve the generalized eigenvalue problem using this PETSc integration. An original attempt was made to integrate Julia directly into the C++ code and solve for the eigenvalue directly using Julia. This workflow was abandoned however when it became clear that the integration was extremely difficult to debug more complicated iterative schemes and provided no real benefit in terms of efficiency. Instead, we made some minor Crunch source-code modifications in order to write these linear operators to disk to be used with various eigensolvers (see Section 3.1).

4.4 Surrogate Model

Concurrent with our Coreform integration efforts, we developed a simplified, surrogate model of a circular PCB that allowed us to continue investigating DAKOTA. The design space was reduced by using symmetry in support positioning. The number of design variables was also reduced by changing from a Cartesian coordinate system (two x-locations and two y-locations) to a reduced polar coordinate system (two radial positions and a separation angle, $\Delta\theta_{1-2}$). A manual pattern search was used to create 32 unique support positions. For a given set of position for the

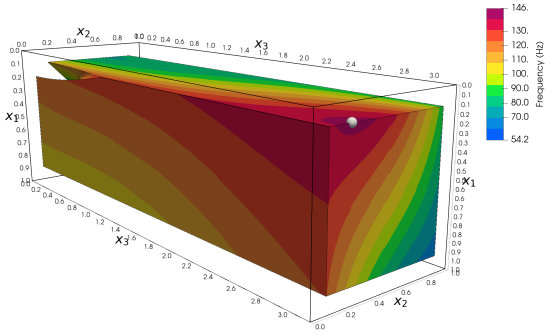


Figure 9: Solution (sphere) returned by `fmincon` superimposed on the feasible solution space of the simple surrogate model.

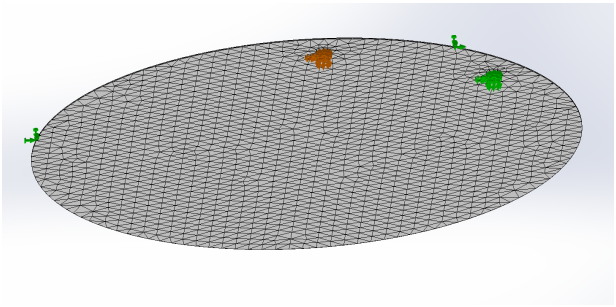


Figure 10: Example of meshed geometry in SolidWorks Simulation used to create the two-support surrogate model

supports SolidWorks Simulation™ was then used to compute the fundamental frequency.

The support positions and resulting fundamental frequencies were tabulated and processed via MATLAB's Curve Fitting Toolbox™ to create a surrogate objective function. We then included the aforementioned nonlinear inequality constraints on support position in relation to the PCB boundaries, as well as an additional constraint that enforced a minimum separation distance between the supports. The surrogate model met the requirements for a solution matching expected behavior.

While rudimentary, the surrogate model gave us insight into the considerations for obtaining frequencies from an FEA package and also led to investigating the efficient global optimizer in DAKOTA.

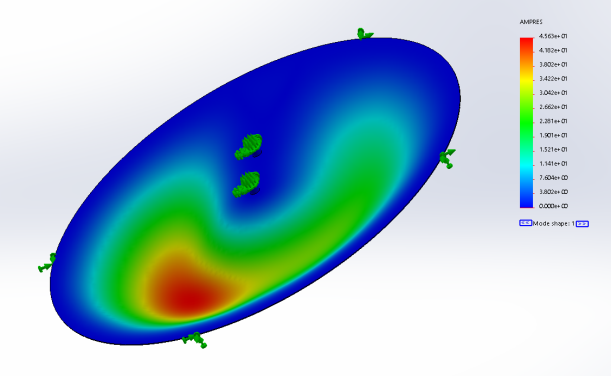


Figure 11: Example results from SolidWorks Simulation. Displacement constraints are located around the rim of the disk and the base of the PCB supports.

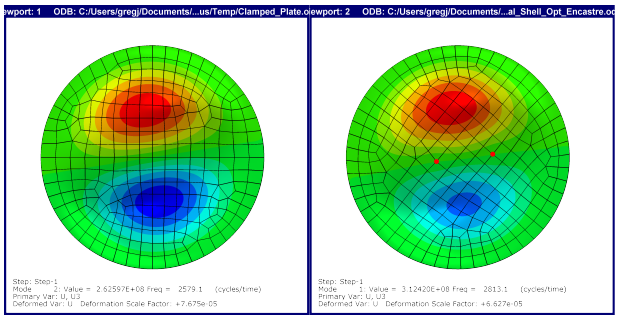


Figure 12: Verification of the optimization results for the disk problem. The left image is of the disk's second mode when no supports are used. Encastring the disk at the locations shown as red dots in the image on the right results removes the first eigenmode from the problem, resulting in the second mode becoming the fundamental mode of the disk.

5 Results

5.1 Optimization on circular membrane

To facilitate testing our DAKOTA-based optimization workflow we began by attempting to optimize support placements on a circular disk with *encastre* boundary conditions[5]. We use the `efficient_global` method in DAKOTA to find the optimal configuration for a two support configuration. In figure 12 we show a post-optimization, verification analysis where we compare the base problem statement against the optimized configuration. We observe that the fundamental frequency of the optimized configuration is $\sim 9\%$ higher than the second mode of the unsupported configuration[6]. Thus we conclude that these optimal locations remove the first vibrational mode and accomplish the task of raising the fundamental frequency.

Further investigation explores an interesting behaviour, which confirms the motivation for pursuing this project. Recall that in a 1D vibrational problem, for a given eigenmode, there exist 0D points where the amplitude is zero - these positions are called the *nodes* of the modeshape. Likewise in higher dimensions there also exist regions of the modeshape where the amplitude is zero; for the N-dimensional unconstrained vibration problem there can exist subspaces of dimension $N-1$, embedded within the N-dimensional space, where the amplitude is zero. For a 2D problem we call this feature a *nodal-line*[10]. Theoretically, a zero-displacement (pinned) boundary condition could be applied along the entire nodal-line(s) of an eigenmode, which would have the effect of removing all lower eigenmodes from the eigenspace, resulting in the eigenmode (and eigenvalue) becoming the fundamental mode. However, as we demonstrate in figure 13, there appear to be subspaces (of dimension $N-2$) of this nodal-line which appear to have the same effect as constraining the entire nodal-line. The engineering value of these positions is that they potentially allow the engineer to minimize the hardware required to maximize the fundamental frequency.

5.2 Optimization on constrained PCB geometry

In figures 14, 15, and 16 we present the optimzation results, as determined via DAKOTA's `efficient_global` method, for the four-, five-, and six-support configurations. These figures demonstrate the efficacy of the optimization framework in determining desirable support locations. An especially interesting behaviour can be seen in figure 16 where we observe that a large increase in the fundamental frequency is obtained. As opposed to placing these supports directly on the nodal-lines, as suggested Ong & Lim[2], for a given eigenmode, instead the supports appear to be making a close approximation to the nodal-lines across *multiple* eigenmodes - therein removing multiple low-frequency eigenmodes. This appears to support the findings of Won & Park[1], who noted that sufficiently stiff supports

[5] Encastre boundary condition: $\Delta x = \Delta y = \Delta z = \Delta\theta_x = \Delta\theta_y = \Delta\theta_z = 0$. This is in contrast to a *pinned* boundary condition: $\Delta x = \Delta y = \Delta z = 0$.

[6] The reason for the higher frequency is that, due to current limitations in Coreform Crunch, we are not able to apply *pinned* boundary conditions, which is the accurate representation of the solution at a node of a modeshape. Instead we are only able to apply *encastre* boundary conditions which is significantly stiffer than the pinned condition

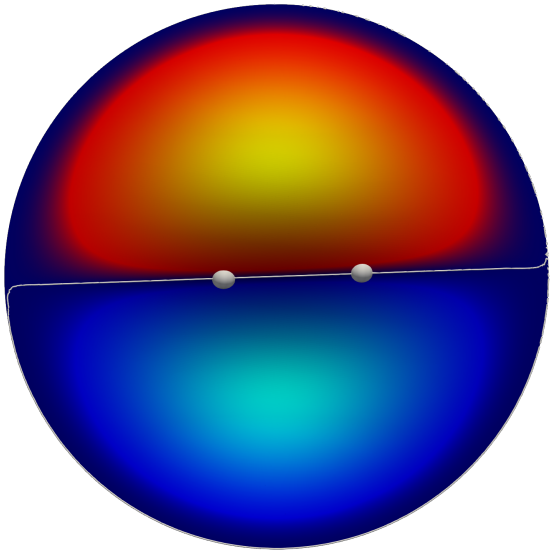


Figure 13: Here we overlay the optimization results for the support locations (white spheres) ontop of the second eigenmode of the unsupported configuration. The thin white line denotes the nodal-line of this eigenmode. This mode corresponds to the first mode of the optimized problem (see figure 12).

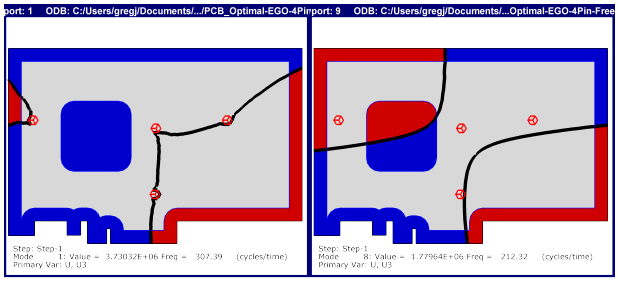


Figure 14: Comparing the results of optimization of a 4-support problem (left), to the *second* non rigid-body mode of the unsupported PCB (right). While the supports are only active in the left case, we provide overlays of their locations on the unsupported cases for reference. Contours depict displacements in the page-normal direction, while nodal-lines are depicted as thick black curves. The feasible region for the supports is the overlaid white geometry.

have a *region* of optimality rather than only being optimal at a single point. While we modeled infinitely-stiff supports, through our encastre boundary conditions, this is likely a valid assumption based on the analysis of Won & Park, who found the critical support stiffness for their *steel plate* problem to be $\sim 10^5 \frac{\text{N}}{\text{m}}$, while a #10-32 10mm PCB mounting screw has a stiffness of $\sim 10^8 \frac{\text{N}}{\text{m}}$.
While we can not confirm whether these positions are in fact the global optimum, by comparing these results with the corresponding eigenmodes of the unsupported PCB, we feel these results are at least satisfyingly close to a global optimum as they either exceed or are close to the eigenvalues of the unsupported PCB. A likely reason for not being able to exactly match the mode shapes, as we were able to observe in the aforementioned disk problem, is that the true globally optimal solution may not lay within the feasible space and that, for both the unconstrained and constrained problems, there exist a large amount of local minima. It is therefore reasonable that these results might reflect a near-optimal solution contained within the feasible domain.

6 Conclusion

An important finding from this study was that gradient-based methods struggle to solve this problem as posed. This is most likely caused by the ever changing mesh required when supports are evaluated at a even slightly different locations adversely impacts the reliability of gradients. Additionally, as this is an extremely multi-modal problem local-solvers by themselves are not reliable for finding globally-optimal values. We found that global, gradient-free methods were essential to robustly obtaining solutions to this problem. However, it stands to reason that improvements to efficiency could be attained reliable gradient-based methods (see future work 6.1).
We've demonstrated a basic framework to automate Coreform's Flex and Crunch software via Sandia's DAKOTA software, especially as it pertains to optimization problems. We've also identified features that could be implemented in Flex & Crunch that would improve its functionality within DAKOTA-based workflows. This work has laid a good foundation for an optimization workflow that can be used by several of Coreform's customers on many different problems moving forward. A by-product of this effort was an effective, albeit rudimentary, approach to determining optimal placements of PCB supports in vibration-critical applications.

6.1 Future Work

This report presents only a very basic DAKOTA/-Coreform workflow with most if not all integration happening manually through file transfer. This is obviously not optimal and there are some very basic things that could be done to improve this. One of the most obvious changes that would be needed

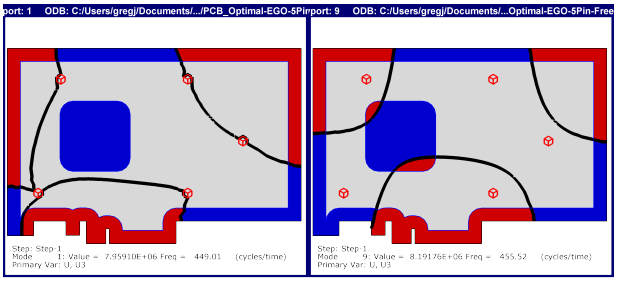


Figure 15: Comparing the results of optimization of a 5-support problem (left), to the *third* non rigid-body mode of the unsupported PCB (right). While the supports are only active in the left case, we provide overlays of their locations on the unsupported cases for reference.

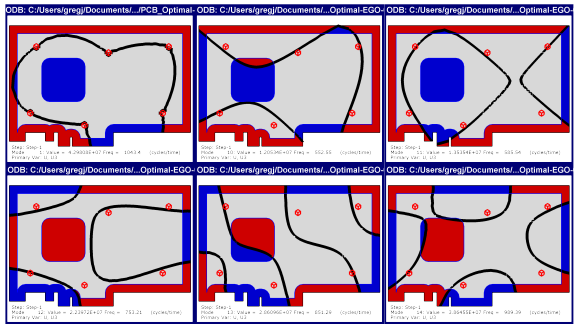


Figure 16: The 6-support case is able to suppress many additional modes. The first mode for the supported problem is shown at top left and then continuing from top-left to bottom-right are the 4th-8th modes of the unsupported PCB. While the supports are only active in the top-left case, we provide overlays of their locations on the unsupported cases for reference.

would be a more standard API for extracting desired results. Currently the only output available in Coreform Crunch is a ParaView output file that is extremely cumbersome to work with outside the ParaView program. In this case the stiffness and mass matrices were used directly, but in general, if the user desired quantities like displacement or stress there is currently no easy way to obtain that information.

As mentioned in section 4.2 a mixed-degree, mixed-continuity, smooth-spline basis was used in the analysis. There is significant outstanding research that could be conducted to understand the impact that using these splines has on this type of frequency analysis. There is also much more work to be done on understanding these vibration-critical applications since only very simple geometries and configurations were considered. To what extent this method produces pragmatic solutions is still largely unknown and additional validation and testing would be vital in supplementing and supporting this work.

It was previously discussed that local gradient-based solvers struggled with even small changes in support locations: when support locations change, a unique mesh created, essentially creating an entirely different problem so computing derivatives via finite differencing is unreliable. An alternative approach would be to enforce the boundary conditions *weakly* instead of strongly. Strong enforcement requires interpolatory nodes in order to function properly which requires the mesh to fit perfectly with support locations. Weak enforcement of boundary conditions would allow a single, simple mesh to be created upfront and then use standard formulations such as Lagrange, penalty, Nitsche etc. to enforce the boundary conditions. This has the potential to drastically improve calculation of gradients as changes in the design variables (i.e. boundary condition locations) would not change the pre-Dirichlet stiffness and mass matrices, while the post-Dirichlet matrices would have predictable, well-behaved scalings. Furthermore weak enforcement would enable a drastic speedup in the objective function calculation, as the pre-Dirichlet matrices would only have to be computed once and could then be reused for every function evaluation.

To generalize the results from a PCB support optimization more parameters would need to be included in the solid modelling. A higher fidelity model would include the stiffness and mass of surface mounted components (e.g. the CPU) and could additionally include optimal placement of some of these components, while leaving some components (e.g. system-level interfaces) fixed.

References

[1] K.-M. Won and Y.-S. Park. Optimal support positions for a structure to maximize its fundamental natural frequency. *Journal of Sound and Vibration*, 213(5):801 – 812, 1998.

[2] J. H. Ong and G. H. Lim. A Simple Technique for Maximizing the Fundamental Frequency of Vibrating Structures . *Journal of Electronic Packaging*, 122(4):341–349, 02 2000.

[3] Dayi Ou and Cheuk Ming Mak. Optimization of natural frequencies of a plate structure by modifying boundary conditions. *The Journal of the Acoustical Society of America*, 142(1):EL56–EL62, 2017.

[4] Kun-Nan Chen. Optimal Support Locations for a Printed Circuit Board Loaded With Heavy Components. *Journal of Electronic Packaging*, 128(4):449–455, 02 2006.

[5] Peter Hagedorn and Anirvan DasGupta. *Vibrations and waves in continuous mechanical systems*. J. Wiley, 2007.

[6] Brian Adams, Michael Eldred, Gianluca Geraci, Russell Hooper, John Jakeman, Kathryn Maupin, Jason Monschke, Ahmad Rushdi, J Stephens, Laura Swiler, and et al. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.11 User's Manual*. Sandia National Laboratories, Nov 2019.

[7] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39):4135–4195, Oct 2005.

[8] Derek Thomas, Luke Engvall, Steven Schmidt, Kevin Tew, and Michael Scott. U-splines: splines over unstructured meshes (pre-print). page 48.

[9] Kyle Richardson, Derek Thomas, and Michael Scott. Stable Time Steps for Explicit Dynamics in Isogeometric Analysis Using U-splines (Preprint). page 20.

[10] Jong-Shyong Wu. *Analytical and numerical methods for vibration analyses*. John Wiley & Sons Singapore Pte. Ltd., 2013.

Appendices

A Problem Specifications

	Disk	**PCB**
K	1×10^6	24×10^3 MPa
ν	0.33	0.33
ρ	1×10^{-5}	1.85×10^{-9} $\frac{\text{tonne}}{\text{mm}^3}$
t	0.025	1.57 mm
r	1.	–

B Geometry Availability

The geometry for the PCB used in this study is provided in the Zenodo distribution of this report, or may also be found in the source-code repository listed in appendix C

1. PCB.stp – STEP file of the PCB midsurface
2. PCB_feasible.stp – STEP file of the feasible region midsurface of the PCB

C Source-Code Availability

The source code for this project is available on GitHub:
https://github.com/GregVernon/Dakota_CFS