forum acusticum 2026
8-12 September · Graz, Austria

JAX-BEM: GRADIENT-BASED ACOUSTIC SHAPE OPTIMISATION VIA A DIFFERENTIABLE BOUNDARY ELEMENT METHOD

James Hipperson[1,2], Jonathan A. Hargreaves[1], Trevor J. Cox[1]

[1] *Acoustics Research Centre, University of Salford, UK*
[2] *Funktion-One Research Ltd. Dorking, UK*

This paper is a revised version of the authors' submitted manuscript that was peer-reviewed by at least two anonymous reviewers.

Abstract

Engineering structures are increasingly designed using numerical optimisation. However, traditional optimisation methods can be challenging with multiple objectives and many parameters. In machine learning, stable training of artificial neural networks with millions or billions of parameters is achieved using automatic differentiation frameworks such as `JAX` and `Pytorch`. Because these frameworks provide accelerated numerical linear algebra with automatic gradient tracking, they also enable differentiable implementations of numerical methods to be built. This facilitates faster gradient-based optimisation of geometry and materials, as well as solution of inverse problems. We demonstrate `JAX-BEM`, a differentiable Boundary Element Method (BEM) solver, showing that it matches the error of existing BEM codes for a benchmark problem and enables gradient-based geometry optimisation. Although the demonstrated examples are for acoustic simulations, the concept could be readily extended to electromagnetic waves.

Keywords: *boundary element method, automatic differentiation, optimization, numerical methods*

1. Introduction

In acoustics, there is a need to characterise radiation from devices such as loudspeakers, scatterers or metamaterials, often in free-field conditions equivalent to an unbounded domain. Domain-based numerical methods, such as the Finite Element Method (FEM) and Finite Difference Time Domain (FDTD), can simulate this using Perfectly Matched Layers (PMLs), infinite elements, or Absorbing Boundary Conditions (ABCs) [1], but the air domain must still be meshed, which creates many degrees of freedom as the number of elements scales with domain size (approximately quadratically in 2D with area and cubically in 3D with volume). The Boundary Element Method (BEM), on the other hand has significant advantages in this use case because it only requires meshing of boundaries. A disadvantage is that BEM requires a dense N^2 matrix, although several mitigating techniques have been developed.

Using automatic differentiation frameworks to build differentiable numerical methods is a recent area of development, although adjoint methods are related [2]. Differentiability enables gradient-based optimisation, inverse problems such as material parameter characterisation, and tight integration with machine learning methods. A differentiable FEM implementation was demonstrated by Xue et al. in 2023 [3] and used for acoustic optimisation by Borrel-Jensen and Bjorgaard in 2025 [4]. In this paper, we demonstrate gradient-based geometry optimisation using JAX-BEM for a loudspeaker horn in an unbounded domain.

2. Background

2.1. Boundary Element Method (BEM)

BEM utilises the Huygens-Fresnel principle and the Kirchhoff-Helmholtz boundary integral to efficiently compute solutions to problems within a domain using only data defined on its boundaries. In unbounded domains, the Sommerfeld radiation condition can be used to show that the contribution from an imagined very distant outer boundary is zero, hence only data on the boundary of the obstacle is required, leading to a very efficient numerical representation.

BEM is formulated in two phases: (1) A system of equations is set up relating the complex amplitudes of a distribution of sources on the boundary to pressure (or its gradient) elsewhere on the boundary S. The boundary conditions are then inserted and this is solved numerically assuming a mesh of elements and interpolation functions. (2) The solution from step 1

***Corresponding author**: j.r.hipperson@edu.salford.ac.uk.*

(being pressure and/or velocity on the boundary) is used to calculate the scattered pressure at points in the domain Ω.
Both stages are based on the Kirchhoff-Helmholtz boundary integral equation:

$$p_\mathrm{s}(\mathbf{r}) = \oint_S \left[G(\mathbf{r},\mathbf{r}') \frac{\partial p_\mathrm{t}(\mathbf{r}')}{\partial n'} - p_\mathrm{t}(\mathbf{r}') \frac{\partial G(\mathbf{r},\mathbf{r}')}{\partial n'} \right] dS', \tag{1}$$

Here p_s is scattered pressure and $p_\mathrm{t} = p_i + p_\mathrm{s}$ is total pressure, where p_i is some incident pressure field that is required if a scattering problem is being modelled. $\mathbf{r} \in \Omega$ and $\mathbf{r}' \in S$ are domain and boundary points and G is the free space Green's function at wavenumber k:

$$G(\mathbf{r},\mathbf{r}') = \frac{\mathrm{e}^{ik|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} \tag{2}$$

Slightly different formulations arise for internal and free-field (exterior) domains and varying boundary conditions. In either case, collocating at N boundary nodes yields a dense system of matrix equations size N^2 to be solved for the unknown boundary quantity. Other variations in formulation exist such as Burton-Miller [5] and CHIEF [6] to address non-uniqueness.

2.2. Automatic Differentiation (autodiff)

The methods that made deep learning possible are backpropagation [7] and automatic differentiation [8]. These are used to train models with millions or billions of parameters [9], which was not feasible with previous optimisation methods. Automatic differentiation works by storing a computational graph of an algorithm step-by-step and simultaneously constructing a second computational graph of derivatives at runtime. This enables gradients to be tracked through an algorithm automatically. Backpropagation (more generally, reverse-mode automatic differentiation) is the process by which gradients are propagated from the end of an algorithm, typically a scalar loss function, backwards through the algorithm with respect to the input parameters using the chain rule. Software frameworks facilitate this, e.g. `JAX` [10] and `Pytorch` [11]. Beyond neural network weights and biases, automatic differentiation can also be used to build differentiable implementations of numerical methods.

3. Methodology

Using the open source BEM project `bempp` [12] as a reference, the operators were refactored to just-in-time (JIT) compiled vectorised `JAX` functions. Because `JAX` can automatically track gradients, the only manual handling of differentiability was to compute the adjacency information of a mesh outside the geometry optimisation loop (as this necessarily contained non-differentiable control flow statements), and to use implicit function theorem (implicit differentiation) to avoid "unrolling" the iterative solver (`GMRES`) [13], which would be inefficient and use a lot of memory.
`JAX` supports accelerated linear algebra operations on CPU, GPU and TPU hardware via the `XLA` backend. This is useful as the boundary-to-domain propagation phase of BEM is "embarrassingly parallel" and highly amenable to GPU and TPU computation.

3.1. Non-differentiable statements

While `JAX` closely follows the functionality of `Numpy` [14], it is recommended to follow a pure function approach, avoid loops and control flow statements. This ensures the `JAX` program is JIT compilable, and differentiable. Loops are problematic as they must be unrolled to track gradients, and loops with an indeterminate number of iterations are not differentiable. Branching statements such as `if` or `switch` are not differentiable and cannot be used within the optimisation loop. Keeping these factors in mind while programming, the automatic handling of gradient tracking in `JAX` makes writing differentiable programs relatively straightforward.
A specific difficulty with BEM is computation of the adjacency matrix and handling of singular integration routines. Special handling is required for element self-interaction (singular) and adjacent element interaction (nearly-singular). Here, branching statements cannot be avoided, so adjacency and selection of integration types is re-computed once per iteration before the differentiable sequence (Alg. 1).

Algorithm 1 Geometry optimisation for scattering

Require: $p_i, mesh_\mathrm{init}, k$
Ensure: $mesh$
  $Vertices, Elements \leftarrow \mathrm{load}(mesh_\mathrm{init})$
  for $iteration = 1$ to K **do**
    if $iteration > 0$ **then**
      $Vertices, Elements \leftarrow \mathrm{load}(mesh)$
    end if
    $p_s, A \leftarrow \mathrm{forward}(k, p_i, Vertices, Elements)$
    $g \leftarrow \mathrm{loss}(p_s)$
    $\nabla_V \leftarrow \mathrm{backward}(p_\mathrm{s}, A, g)$
    $vertices \leftarrow vertices - H^{-1} \cdot \nabla_V$
    $mesh \leftarrow vertices$
  end for
  return $mesh$

Here A is the boundary integral operator matrix (left hand side), $mesh_\mathrm{init}$ is the initial/starting mesh, and H^{-1} is the inverse Hessian approximated by the L-BFGS optimiser [15]. Forward is shown in Alg. 2 and backward in Alg. 3.

3.2. Implicit differentiation

`GMRES` is an iterative solver and this is problematic in differentiable computing for two reasons, it is: (1) iterative and runs in a loop, and (2) typically configured to stop at a target tolerance, which means that the number of iterations is indeterminate. `JAX` contains several useful tools to overcome these problems. Using `JAX.custom_vjp` (Vector-Jacobian Product) we define the forward (primal) function and the backward function (pullback) to set up *implicit* differentiation

so that autodiff bypasses both instances of `GMRES`. `nondiff_argnames` excludes non-differentiable arguments from gradient computation by treating them as constants. The forward function is:

Algorithm 2 BEM Forward function

Require: $k, p_\mathrm{i}, Vertices, Elements$
Ensure: p_s, A
  $A \leftarrow \mathrm{assemble}(k, Vertices, Elements)$
  $b \leftarrow \mathrm{project}(p_\mathrm{i})$
  solve $Ax = b$ using GMRES
  $p_\mathrm{s} \leftarrow x$
  return p_s, A

The backward function (Alg. 3) is where implicit differentiation happens, yielding gradients with respect to A and b, and by the chain rule V (vertices). It does this by solving the adjoint system $A^H\lambda = g$. (Note, this is essentially an automated version of the adjoint state method [2].)

Algorithm 3 Implicit `GMRES` Backward function

Require: p_s, A, loss g
Ensure: Gradient ∇_V
  solve $A^H\lambda = g$ using GMRES
  $\nabla_A \leftarrow -\lambda p_\mathrm{s}^H$
  $\nabla_b \leftarrow \lambda$
  $\nabla_V \leftarrow (\partial A/\partial V)^\top \nabla_A$
  return ∇_V

The key is that by using implicit differentiation, autodiff never touches the iterative solver; it remains a black box. Only the inputs and outputs pass gradients. Any solver could be used in principle, but the `JAX` implementation of `GMRES` was used (`jax.scipy.sparse.linalg.gmres`) to enable JIT compilation, as both forward and backward functions are run many times in an optimisation loop.

3.3. Geometry optimisation

The parameters to be optimised are the control points of splines that define the axial horn profile. The initial mesh is then deformed to follow the splines. This work extends prior approaches [16], [17], [18], [19], [20] by using gradients in the optimisation. The L-BFGS optimiser was used, specifically `scipy.opimize.minimize`.

3.4. Loss (cost) function

The loss function is a mean squared error loss against dB targets relative to the normalised on-axis response. This was set to $T_\mathrm{in} = -3$dB (Target inside coverage) ($\pm 35°$ Horizontal, $\pm 25°$ Vertical) and $T_\mathrm{out} = -10$dB (Target outside coverage).

$$\mathcal{L}_\mathrm{MSE} = \frac{1}{|\Theta_\mathrm{in}|F} \sum_{f,\,\theta\in\Theta_\mathrm{in}} \left(D^{(f,\theta)} - T_\mathrm{in}\right)^2 + \frac{1}{|\Theta_\mathrm{out}|F} \sum_{f,\,\theta\in\Theta_\mathrm{out}} \left(D^{(f,\theta)} - T_\mathrm{out}\right)^2 \tag{3}$$

The MSE between the $D^{(f,\theta)}$, being the sound pressure level normalised to on-axis, and the target is evaluated for the in-coverage region and out-of-coverage region for the horizontal and vertical planes separately.

4. Results

4.1. Validation - Error

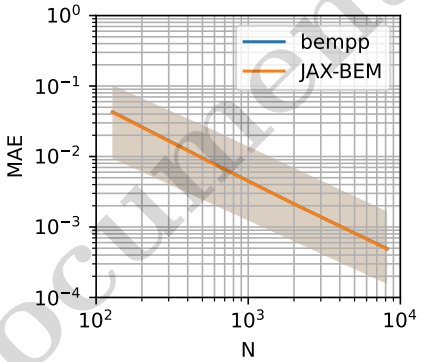

Figure 1: Error vs. analytic solution vs. mesh refinement. 50$^\mathrm{th}$ percentile. Shaded 10$^\mathrm{th}$ to 90$^\mathrm{th}$ percentile range.

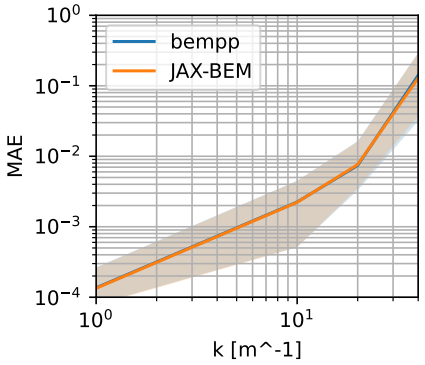

Figure 2: Error vs. analytic solution vs. wavenumber k. 50$^\mathrm{th}$ percentile. Shaded 10$^\mathrm{th}$ to 90$^\mathrm{th}$ percentile range.

The scattering from a rigid sphere in 3D was chosen as a validation problem, because it has analytic solutions in the form of the Mie series [21]. The Burton-Miller formulation was used to avoid spurious interior resonances. Validation of `JAX-BEM` was performed by first computing the analytic solution on a 128^3 grid as a reference, then computing the mean absolute error of solutions from `JAX-BEM` and `bempp`. Mean absolute error compared to analytic solutions ranged from 4×10^{-2} ($N = 128$) to 4×10^{-4} ($N = 8192$) for both `JAX-BEM` and `bempp`, demonstrating good agreement (Figs. 1, 2).

Although `bempp` uses `numpy` which defaults to 64 bit precision (`complex128`) there was no significant difference in error to `JAX` using the default 32 bit precision (`complex64`), indicating that discretisation error dominates.

4.2. Wall clock time

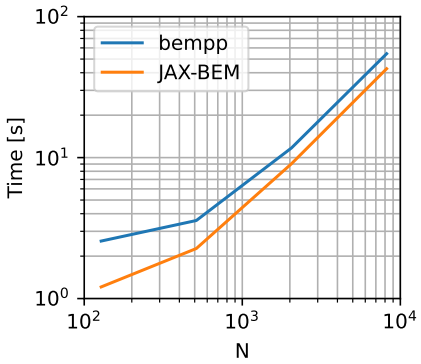

Figure 3: Wall clock time for varying mesh refinement N and $k = 4$. `bempp` using `numba` JIT backend. `JAX` without calculation of gradients.

Fig. 3 shows the wall clock time vs number of elements N. There were 2×10^6 domain evaluation points on a 128^3 3D grid. The CPU was an AMD Ryzen™ AI Max+ 395. `bempp` is using the `numba` JIT backend, while `JAX-BEM` uses the built-in `XLA` backend. For both methods, timing was performed after a small initial warm-up solve to allow the JIT functions to compile. `JAX-BEM` likely performs slightly better due to its dense assembly, `bempp` on the other hand uses a chunked approach.

4.3. Geometry optimisation

We use reverse-mode automatic differentiation to optimise the directivity of a loudspeaker horn driven by constant unit velocity. A coupled formulation was used, consisting of a closed interior domain (horn) with rigid boundary conditions, radiating throat cap and fictitious mouth cap. The interior domain is terminated with an absorbing Dirichlet-to-Neumann map boundary condition at its mouth, data from which is fed into the unbounded exterior domain model. It is noted that other formulations may be more accurate. Loudspeaker horns are sensitive and difficult to design for wide bandwidth constant directivity. Therefore, gradient-based optimisation is particularly attractive for this application. For the optimisation, the target directivity was set to 70° horizontal, 50° vertical. The solver ran in batches of 20 frequencies logarithmically spaced from 4 kHz to 18 kHz. The exterior domain was sampled using three arcs of observation points at 10 m spaced at 1° intervals across 90° in the horizontal, vertical and 45° planes.

Observing the differences between the initial mesh (Fig. 4) and the optimised mesh (Fig. 5) several interesting features are apparent. The most significant

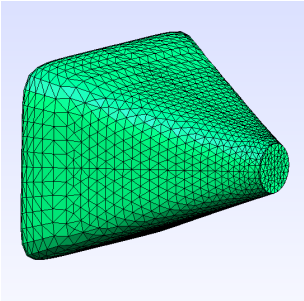

Figure 4: Intitial horn mesh.

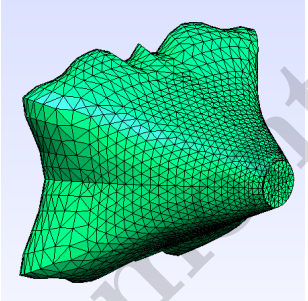

Figure 5: Optimised horn mesh.

is that the optimiser has arrived at a complex mouth shape. A smooth mouth termination has been added, and this varies from a very large flare at 45° to a sharper termination in the horizontal and vertical planes. A smooth edge termination contributes the most to the improvements seen in the directivity plot. Fig. 6 shows the normalised horizontal and vertical directivity maps of an unoptimised design (simple conical horn). An approximately constant directivity characteristic is apparent, but there is substantial diffraction above 8 kHz in both planes.

Fig. 7 shows the directivity plots of the optimised design, the diffraction has been reduced, and the general characteristic is that of more constant directivity in the horizontal plane. However, in the vertical plane, although the directivity has been widened to the target, the out of coverage sound pressure level has increased at some frequencies, indicating the difficulty of competing optimisation objectives.

Although the mesh was small (a quadrant was used with symmetry planes, 948 elements) the memory usage was approximately 100 GB for 20 frequencies per iteration in 32 bit precision (`complex64`). The current implementation is by no means fully optimised, but this does highlight a fundamental difficulty with BEM in that it scales as $\mathcal{O}(N^2)$. H-Matrix compression [22] is a potential solution, if implemented in a differentiable framework.

5. Discussion and further work

Validation of the forward algorithm and an optimisation example have been demonstrated. The most important area for further work is to rigorously validate the accuracy of gradients, as optimisers can still work with noisy or slightly inaccurate gradients.

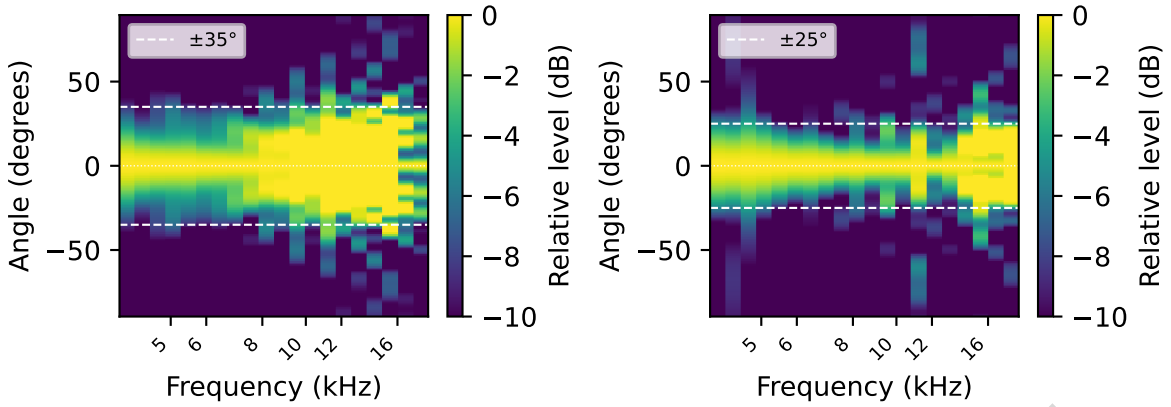

Figure 6: Initial directivity (normalised to on-axis). Left: Horizontal, Right: Vertical.

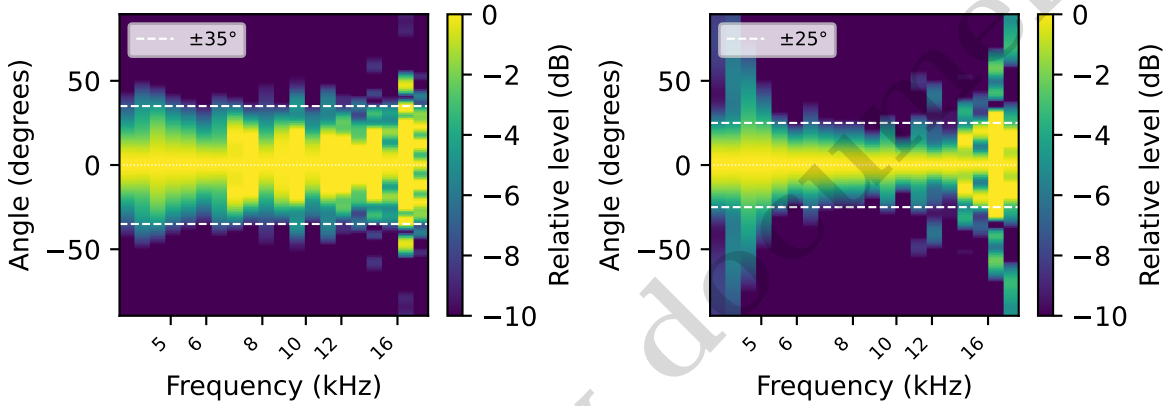

Figure 7: Optimised directivity (normalised to on-axis). Left: Horizontal, Right: Vertical.

Multiple full BEM solves (several frequencies) per iteration is very expensive, and automatic gradient tracking increases memory usage. Therefore, a key avenue for further work is to explore methods by which a full re-solve can be avoided for each geometry update, or at least avoiding instantiating the full N^2 operator matrix. Because each optimisation iteration is an incremental change to the mesh, it seems inefficient that the entire problem must be solved from scratch. Warm-starting `GMRES` with the previous solution provides a useful speed-up, but the forward algorithm remains expensive.

There are several possible BEM formulations to model a horn radiating into an unbounded domain. Gradient-based optimisation and implicit differentiation are new parameters that influence which choice of formulation is the most accurate and efficient. It was found that a Dirichlet-to-Neumann map formulation was more stable than a coupled subdomain formulation. This should be examined in more detail.

Validation with acoustic measurements of 3D printed devices should also be performed. This should be combined with a comparison to conventional (non-gradient-based) optimisation in terms of performance and computation time.

6. Conclusions

We have demonstrated a fast, differentiable and GPU-computable boundary element method implemented using the automatic differentiation framework `JAX`. The method is differentiable from the domain solution back to the mesh vertices, computing gradients of the loss function with respect to the input mesh geometry. This enables efficient optimisation of many input parameters using a scalar loss value with reverse-mode automatic differentiation. The approach was applied to the problem of optimising loudspeaker horn directivity.

7. Code

The `JAX-BEM` code and validation example (scattering from a rigid sphere) can be downloaded from `https://github.com/jrhip/jax-bem`

8. Acknowledgments

This work was supported by EPSRC and Funktion-One Research Ltd. as part of the CDT in Sustainable Sound Futures. (https://soundfutures.salford.ac.uk/)

References

[1] T. Sakuma, S. Sakamoto, and T. Otsuru, Eds., *Computational Simulation in Architectural and Environmental Acoustics: Methods and Applications of Wave-Based Computation*, 1st ed. Tokyo: Springer, 2014, ISBN: 978-4-431-54453-1. DOI: 10.1007/978-4-431-54454-8.

[2] J. Cea, "Conception optimale ou identification de formes, calcul rapide de la dérivée directionnelle de la fonction coût," fr, *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 20, no. 3, pp. 371–402, 1986. DOI: 10.1051/m2an/1986200303711.

[3] T. Xue et al., "JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science," en, *Computer Physics Communications*, vol. 291, p. 108 802, Oct. 2023. DOI: 10.1016/j.cpc.2023.108802.

[4] N. Borrel-Jensen and J. Bjorgaard, *Differentiation Strategies for Acoustic Inverse Problems: Admittance Estimation and Shape Optimization*, en, Nov. 2025. DOI: 10.48550/arXiv.2511.11415.

[5] A. J. Burton and G. F. Miller, "The application of integral equation methods to the numerical solution of some exterior boundary-value problems," en, *Proc. Royal Society of London. A. Mathematical and Physical Sciences*, vol. 323, no. 1553, pp. 201–210, Jun. 1971. DOI: 10.1098/rspa.1971.0097.

[6] H. A. Schenck, "Improved Integral Formulation for Acoustic Radiation Problems," en, *The Journal of the Acoustical Society of America*, vol. 44, no. 1, pp. 41–58, Jul. 1968. DOI: 10.1121/1.1911085.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," en, *Nature*, vol. 323, pp. 533–536, 1986.

[8] A. Griewank and A. Walther, *Evaluating Derivatives*, Second. Society for Industrial and Applied Mathematics, 2008. DOI: 10.1137/1.9780898717761.

[9] T. B. Brown et al., *Language Models are Few-Shot Learners*, en, Jul. 2020. DOI: 10.48550/arXiv.2005.14165.

[10] J. Bradbury et al., *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: http://github.com/jax-ml/jax.

[11] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," *CoRR*, vol. abs/1912.01703, 2019. arXiv: 1912.01703.

[12] T. Betcke and M. Scroggs, "Bempp-cl: A fast Python based just-in-time compiling boundary element library.," en, *JoSS*, vol. 6, no. 59, p. 2879, Mar. 2021. DOI: 10.21105/joss.02879.

[13] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," en, *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, Jul. 1986. DOI: 10.1137/0907058.

[14] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2.

[15] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," en, *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, Aug. 1989. DOI: 10.1007/BF01589116.

[16] R. Udawalpola, E. Wadbro, and M. Berggren, "Optimization of a variable mouth acoustic horn," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 5, pp. 591–606, Feb. 2011. DOI: 10.1002/nme.2982.

[17] L. Godinho, P. Amado-Mendes, J. Carbajo, and J. Ramis-Soriano, "3D numerical modelling of acoustic horns using the method of fundamental solutions," en, *Engineering Analysis with Boundary Elements*, vol. 51, pp. 64–73, Feb. 2015. DOI: 10.1016/j.enganabound.2014.09.013.

[18] S. Schmidt, E. Wadbro, and M. Berggren, "Large-Scale Three-Dimensional Acoustic Horn Optimization," en, *SIAM Journal on Scientific Computing*, vol. 38, no. 6, B917–B940, Jan. 2016. DOI: 10.1137/15M1021131.

[19] P. R. Andersen, V. Cutanda Henríquez, N. Aage, and J. Kook, "3D shape optimization of loudspeaker cabinets for uniform directivity," en, *Structural and Multidisciplinary Optimization*, vol. 65, no. 12, p. 343, Dec. 2022. DOI: 10.1007/s00158-022-03451-2.

[20] H. Dong, J.-B. Doc, and S. Félix, "Efficient multimodal-based shape optimization of acoustic horns with application to subwavelength perfect transmission," en, *Journal of Sound and Vibration*, vol. 559, p. 117 746, Sep. 2023. DOI: 10.1016/j.jsv.2023.117746.

[21] P. M. Morse and K. U. Ingard, *Theoretical Acoustics*. New York: McGraw-Hill, 1968.

[22] K. Bruyninckx, D. Huybrechs, and K. Meerbergen, "Uniform $\mathcal{H}$-matrix compression with applications to boundary integral equations," *SIAM Journal on Scientific Computing*, vol. 47, no. 3, B618–B644, May 2025. DOI: 10.1137/24m1665209.