Engineering with Computers
https://doi.org/10.1007/s00366-025-02107-1

ORIGINAL ARTICLE

Helmholtz-x : Parallelized adjoint open source solver for the thermoacoustic Helmholtz equation

Ekrem Ekici[1] · Stefano Falco[1] · Matthew P. Juniper[1]

Received: 13 May 2024 / Accepted: 10 January 2025

Abstract
We create and describe an inhomogeneous Helmholtz equation solver, helmholtz-x, written in an open-source framework. The mesh is generated with Gmsh and the solver uses DOLFINx and UFL from FEniCSx. The performance, validity, stability and extensibility of the solver are demonstrated through several examples of thermoacoustic instability, from the one-dimensional Rijke tube to the three-dimensional MICCA combustor. The implementation of Bloch-type boundary conditions is explained and tested. The adjoint capability of the solver is also shown, and used to obtain derivatives of the eigenvalue with respect to shape parameters. This is exploited to find shape changes that reduce the thermoacoustic growth rate.

Keywords Helmholtz equation · Finite element method · Open-source software · Adjoint · Parallel computing

1 Introduction

Thermoacoustic oscillations in rockets and gas turbines occur when the fluctuating heat release rate occurs sufficiently in phase with the acoustic pressure that the growth of acoustic energy exceeeds the damping [1]. These oscillations damage engines and must be eliminated, preferably at the design stage.

Low-order network models such as LOTAN [2], OSCILOS[1] [3] and taX[2] [4] are the simplest tools for modelling thermoacoustic instability. In these models, the acoustics are modelled within connected modules with simple geometries and the temperature is assumed uniform within each module. The heat release rate from the flame is typically modelled as a compact source of heat, which is a function of the acoustic velocity or pressure. Low-order network models can only model simple geometries but can account for mean flow effects such as entropy waves.

For complicated geometries and spatially-varying temperature fields, finite element method (FEM) can be used to solve the non-homogenous Helmholtz equation. These Helmholtz solvers assume that the mean flow Mach number is small and therefore cannot model entropy waves. For example, the package PyHoltz[3] is a Python-based FEM solver that calculates thermoacoustic eigenvalues and corresponding eigenvectors. It includes Bloch boundary conditions [5, 6], uncertainty quantification [7] and non-iterative solvers [8]. It has been re-written in Julia under the name WavesAndEigenvalues[4]. These packages implement nonlinear eigenproblem solvers with subspace [9] and iterative algorithms such as Banach' fixed point iteration and Householder's method [10]. These packages also have some adjoint capability [11].

For resolution of the reacting flow within a combustion system, Large Eddy Simulation (LES) can be used, but at high computation cost. This provides high fidelity information about the flow, which can be used to extract acoustics through, for example, dynamic mode decomposition [12]. LES is used to investigate self-excited thermoacoustic instabilities for laboratory [13] and industrial [14] combustion systems. However, LES is too expensive for extensive para-

[1] https://github.com/MorgansLab/OSCILOS_long.
[2] https://gitlab.lrz.de/tfd/tax.

✉ Matthew P. Juniper
mpj1001@cam.ac.uk

Ekrem Ekici
ee331@cam.ac.uk

Stefano Falco
sf620@cantab.ac.uk

[1] Engineering Department, University of Cambridge, Trumpington Street, CB21PZ Cambridge, UK

[3] https://bitbucket.org/pyholtzdevelopers/public/src/master/.
[4] https://github.com/JulHoltzDevelopers/WavesAndEigenvalues.jl.

Published online: 07 February 2025

metric studies and its results do not show how to control thermoacoustic instabilities [15].

The stability of thermoacoustic systems is highly sensitive to small changes in many parameters, particularly those that affect the phase between the heat release rate and the acoustic pressure such as the flame configuration [16]. Knowing the thermoacoustic response of the system to these changes would help the design process. Adjoint methods achieve this at a low computation cost. This was first presented in [17] in which stabilizing mechanisms for a hot wire Rijke tube were quickly determined. Adjoints were then applied with a wave-based approach [18] and a Helmholtz solver [19] to determine thermoacoustic sensitivities [20]. Adjoint methods were then used in low-order network models to stabilize longitudinal [21] and annular [22] combustors by changing their shapes. For more complex geometries, adjoint Helmholtz solvers are required, which increases the computational cost. For these geometries, the shapes need to be parametrized with, for example, B-Splines [23] and Non-Uniform B-Splines [24], or more descriptive CAD tools for industrial geometries. Adjoint methods then provide the sensitivity of the thermoacoustic eigenvalues to changes in the shape parameters.

Free-form deformation (FFD) places control points within the volume that is to be deformed and then shifts their positions. The geometry deforms as it moves with the control points [25]. FFD has many design applications, such as aerodynamics [26, 27] and turbomachinery [28–30] but it has not yet been used in thermoacoustics.

In this paper, we present an open-source parallelized adjoint Helmholtz solver, helmholtz-x, which combines several open source packages to model small amplitude thermoacoustic oscillations in complex geometries. In Sect. 2, we derive the numerical system for direct and adjoint frameworks, and explain the implementation of various boundary conditions. In Sect. 3 we verify the results of the solver and show its parallel computation capability. In Sect. 4 we demonstrate adjoint-based shape optimization with FFD.

2 helmholtz-x

In this section, we present the derivation and FEM discretization of the thermoacoustic Helmholtz equation and corresponding boundary conditions. We include code and show how to calculate eigenmodes. helmholtz-x uses the open-source FEM framework, FEniCSx and numerical toolkits PETSc and SLEPc.

2.1 FEniCSx, PETSc and SLEPc

The FEniCSx project [31] offers a framework for solving PDEs using FEM. It has efficient matrix assembly kernels for reducing the solution time. The software also offers a scalable framework for computationally demanding problems with MPI [32] parallelization. It also has a Python interface named mpi4py [33].

We define weak forms of the PDEs through a high-level Python interface with the Unified Form Language (UFL) package [34]. UFL weak forms are used to define the weak forms, which are then used by the FEniCSx Form Compiler, FFCx [35], which generates the low level C codes for local tensors to be globally assembled by DOLFINx. Subsequently, the UFL compiled forms can be assembled as sparse matrices. Matrices can be formatted with the Portable Extensible Toolkit for Scientific Computation (PETSc) [36] so as to be compatible with MPI and to use a Python binding, petsc4py [37]. It also uses the open-source scalable and flexible toolkit for the solution of eigenvalue problems (SLEPc) [38], which solves eigenvalue problems of PETSc matrices, returning eigenvalues and their corresponding PETSc eigenvectors. SLEPc also offers Python binding through slepc4py [37]. FEniCSx, PETSc and SLEPc all support complex numbers.

2.2 Thermoacoustic Helmholtz equation

The derivation of the direct and adjoint thermoacoustic Helmholtz equations follows the methodology in [39]. The direct Helmholtz equation and momentum equation in a domain $\Omega \subset \mathbb{R}^3$ are

$$\nabla \cdot \left(c^2 \nabla \hat{p}_1\right) + \omega^2 \hat{p}_1 = i\omega(\gamma - 1)\hat{q}_1 + c^2 \nabla \cdot \hat{f}_1 + c^2 i\omega \hat{m}_1 \quad \text{in } \Omega, \tag{1a}$$

$$-i\rho_0 \omega \hat{\mathbf{u}}_1 + \nabla \hat{p}_1 = \hat{f}_1 \quad \text{in } \Omega, \tag{1b}$$

where c is the spatially-varying speed of sound, $\hat{p}_1$ is the direct acoustic pressure, $\hat{\mathbf{u}}_1$ is the acoustic velocity, ω is the complex valued angular frequency, γ is the heat capacity ratio, $\hat{q}_1$ is any fluctuating heat release rate, $\hat{f}_1$ is any fluctuating body force, $\hat{m}_1$ is any fluctuating mass injection, and p_0 is the mean pressure. Equation (1) can be written as $\mathcal{L}(\omega)\hat{p}_1 = 0$, where $\mathcal{L}$ is a differential operator that is linear in $\hat{p}_1$ but potentially nonlinear in ω. The property (Sect. 3.1 in [39])

$$\langle \hat{p}_1^\dagger | \mathcal{L}\hat{p}_1 \rangle = \langle \mathcal{L}^\dagger \hat{p}_1^\dagger | \hat{p}_1 \rangle + \text{boundary terms} = 0$$

defines the adjoint Helmholtz and momentum equations [39] as

$$\nabla \cdot \left(c^2 \nabla \hat{p}_1^\dagger\right) + \omega^{*2} \hat{p}_1^\dagger = i\omega^*(\gamma - 1)\hat{q}_1(\omega^*) + c^2 \nabla \cdot \hat{f}_1 + i\omega^* c^2 \hat{m}_1 \quad \text{in } \Omega, \tag{2a}$$

$$-i\rho_0 \omega^* \hat{\mathbf{u}}_1 + \nabla \hat{p}_1^\dagger = \hat{f}_1 \quad \text{in } \Omega, \tag{2b}$$

where $\hat{p_1}^\dagger$ is the adjoint acoustic pressure and ω^* is the complex conjugate of the angular eigenfrequency.

2.3 Source terms in the Helmholtz equation

We assume that the local heat release rate perturbation, q_1, is proportional to the acoustic velocity at a measurement point:

$$\frac{q_1(\mathbf{x},t)}{q_0} = \mathrm{FTF}\frac{\mathbf{u}_1(x_r)\cdot\mathbf{n}_r}{\mathbf{u}_b}, \tag{3}$$

where q_0 is the mean heat release rate, FTF is the complex-valued flame transfer function, which depends on ω, u_b is the mean velocity and $\mathbf{n}_r$ is the unit normal vector in the reference direction. The fluctuating heat release rate, $\hat{q}_1$, is often modelled with a local $n-\tau$ formulation [40]. In Eq. (4), as in [19]:

$$\frac{q_1(\mathbf{x},t)}{q_0} = \frac{nh(\mathbf{x})\int_\Omega w(\mathbf{x})\mathbf{u}_1(\mathbf{x},t-\tau(\mathbf{x}))\cdot\mathbf{n}_r\mathbf{dx}}{u_b}, \tag{4}$$

where n is the interaction index, $\tau(\mathbf{x})$ is the time delay, $h(\mathbf{x})$ is the heat release rate distribution and $w(\mathbf{x})$ is the measurement field. In the frequency domain, we can write Eq. (4) as

$$\hat{q}_1 = ne^{i\omega\tau}\int_\Omega \frac{q_0}{u_b}h(\mathbf{x})w(\mathbf{x})\hat{\mathbf{u}}_1(\mathbf{x})\cdot\mathbf{n_r}\mathbf{dx}. \tag{5}$$

The fields $h(\mathbf{x})$ and $\tau(\mathbf{x})$ can be obtained from experiments or simulations [13]. If $\tau(\mathbf{x})$ is uniform, we replace $ne^{i\omega\tau}$ with a complex-valued FTF. Without a fluctuating body force, Eq. (1b) becomes $\nabla\hat{p}_1 = i\omega\rho_0\hat{\mathbf{u}}_1$, so Eq. (5) becomes:

$$\hat{q}_1 = \mathrm{FTF}\frac{q_0}{u_b}h(\mathbf{x})\int_\Omega \frac{w(\mathbf{x})}{i\omega\rho_0}\nabla\hat{p}_1\cdot\mathbf{n_r}\,\mathbf{dx}. \tag{6}$$

We then obtain the thermoacoustic Helmholtz equation with a distributed measurement function:

$$\begin{aligned}&\nabla\cdot\left(c^2\nabla\hat{p}_1\right)+\omega^2\hat{p}_1\\&=\mathrm{FTF}(\gamma-1)\frac{q_0}{u_b}h(\mathbf{x})\int_\Omega\frac{w(\mathbf{x})}{\rho_0(\mathbf{x})}\nabla\hat{p}_1\cdot\mathbf{n_r}\,\mathbf{dx}\quad\text{in }\Omega.\end{aligned} \tag{7}$$

If w is a Dirac delta function, δ_D, then Eq. (1) becomes:

$$\begin{aligned}&\nabla\cdot\left(c^2\nabla\hat{p}_1\right)+\omega^2\hat{p}_1\\&=\mathrm{FTF}(\gamma-1)\frac{q_0}{u_b}h(\mathbf{x})\frac{\nabla\hat{p}_1(\mathbf{x_r})\cdot\mathbf{n_r}}{\rho_0(\mathbf{x_r})}\quad\text{in }\Omega.\end{aligned} \tag{8}$$

We label the case where the FTF (i.e. n) is zero as the *passive flame* and the others as the *active flame*.

2.4 Finite element formulation

We start by defining the Sobolev space $H^1(\Omega)$ as:

$$H^1(\Omega) = \{u\in L^2(\Omega)|\nabla u\in L^2(\Omega)\},$$

where u is any square integrable function and L^2 is the space of square-integrable functions in Ω. We define the properties of L^2 such that

$$\langle u|v\rangle = \int_\Omega uv^*\,dx,$$

and $\langle u|u\rangle \geq 0$. To approximate the solution numerically, we define the test function $v\in V$ in the finite-dimensional function space $V^h\subset H^1(\Omega)$ as

$$V_h = \{v_h\in H^1(\omega)|v_h|_K\in\mathcal{P}_k(K)\forall K\in\mathcal{T}_h\},$$

where $\mathcal{P}_k(K)$ is the space of polynomials degree $\leq$ k on each element K (triangle for 2D and tetrahedra for 3D). We then define $\hat{p}_{1,h}\in V^h$ such that $\hat{p}_{1,h}=\sum_k\phi_k p_{1,k}$ where ϕ_k are real valued basis functions of space V^h and $p_{1,k}\in\mathbb{C}$ are complex valued degrees of freedom.

2.4.1 Discretization

Within the finite element framework, we integrate the terms in (1a) over the domain and multiply by a test function v_h to obtain

$$\begin{aligned}&\int_\Omega\nabla\cdot\left(c^2\nabla\hat{p}_{1,h}\right)v_h\,\mathbf{dx}+\int_\Omega\omega^2\hat{p}_{1,h}\,v_h\,\mathbf{dx}\\&=\int_\Omega i\omega(\gamma-1)\hat{q}_1\,v_h\,\mathbf{dx}+\int_\Omega c^2\nabla\cdot\hat{f}_1\,v_h\,\mathbf{dx}\\&\quad+\int_\Omega c^2 i\omega\hat{m}_1\,v_h\,\mathbf{dx}\;\forall v_h\in V_h.\end{aligned} \tag{9}$$

Note that the final two terms in Eq. (9) are only integrated over the two domains in which the fluctuating body force $\hat{f}_1$ and fluctuating mass $\hat{m}_1$ act, which are assumed to be negligible in this paper. Using integration by parts to reduce the

smoothness requirements of the first term in Eq. (9) yields:

$$\sum_k \left(\sum_j \left(-\int_\Omega c^2 \nabla\phi_k \cdot \nabla\phi_j \, \mathbf{dx} + \int_{\partial\Omega} c^2 \nabla\phi_k \cdot \mathbf{n} \, \phi_j \, \mathbf{dS} + \int_\Omega \omega^2 \phi_k \, \phi_j \, \mathbf{dx} \right) p_{1,k} \right) = \sum_k \left(\sum_j \left(\int_\Omega i\omega(\gamma - 1)\hat{q}_1 \phi_j \, \mathbf{dx} \right) \right), \tag{10}$$

where $\mathbf{n}$ is the normal vector of the relevant boundary. The specific acoustic impedance [41], Z, is defined as

$$Z = \frac{\hat{p}_1}{\rho_0 c \hat{\mathbf{u}}_1 \cdot \mathbf{n}}. \tag{11}$$

Using Eq. (11), we can transform the second integral in Eq. (10) into the Robin integral using Eq. (1b) by writing

$$\int_{\partial\Omega} c^2 \left(\nabla\phi_k \cdot \mathbf{n}\right) \phi_j \, dS = \int_{\partial\Omega} c^2 \left(\frac{i\omega}{cZ} \phi_k \right) \phi_j \, dS. \tag{12}$$

Hence, the matrix form of Eq. (10) is

$$\left[\mathbf{A} - \mathbf{D}(\omega) + \omega\mathbf{B} + \omega^2 \mathbf{C}\right]\mathbf{p} = 0, \tag{13}$$

where

$$\mathbf{A} = -\int_\Omega c^2 \nabla\phi_k \cdot \nabla\phi_j \, \mathbf{dx}, \tag{14a}$$

$$\mathbf{B} = \int_{\partial\Omega} \frac{ic}{Z} \phi_k \, \phi_j \, \mathbf{dS}, \tag{14b}$$

$$\mathbf{C} = \int_\Omega \phi_k \, \phi_j \, \mathbf{dx}, \tag{14c}$$

$$\mathbf{D} = \mathrm{FTF}\,(\gamma - 1)\frac{q_0}{u_b} \int_\Omega \phi_j \, h(\mathbf{x}) \, \mathbf{dx} \int_\Omega \frac{w(\mathbf{x})}{\rho_0} \nabla\phi_k \cdot \mathbf{n}_r \mathbf{dx}, \tag{14d}$$

and $\mathbf{p}$ is the direct eigenvector. In Eq. (14d), there is an outer product between the left integral and the right integral. We denote matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ the *acoustic* matrices and matrix $\mathbf{D}$ as the *flame* matrix.

To derive the adjoints of Eq. (14) in matrix form, we take the conjugate transpose (Hermitian, $(^H)$) of Eq. (14) and calculate the right eigenvector, which is the adjoint eigenvector:

$$\left[\mathbf{A}^H - (\mathbf{D}(\omega))^H + \omega^* \mathbf{B}^H + \omega^{*2} \mathbf{C}^H\right]\mathbf{p}^\dagger = 0, \tag{15}$$

where

$$\mathbf{A}^H = -\int_\Omega c^2 \nabla\phi_j \cdot \nabla\phi_k \, \mathbf{dx}, \tag{16a}$$

$$\mathbf{B}^H = \int_{\partial\Omega} \frac{ic}{Z^*} \phi_j \cdot \phi_k \, \mathbf{dS}, \tag{16b}$$

$$\mathbf{C}^H = \int_\Omega \phi_j \cdot \phi_k \, \mathbf{dx}, \tag{16c}$$

$$\mathbf{D}^H = \mathrm{FTF}^*(\gamma - 1)\eta \int_\Omega \frac{w(\mathbf{x})}{\rho_0} \nabla\phi_j \cdot \mathbf{n}_r \mathbf{dx} \int_\Omega \phi_k \, h(\mathbf{x}) \mathbf{dx}. \tag{16d}$$

and $\mathbf{p}^\dagger$ is the adjoint eigenvector. Matrices $\mathbf{A}$ and $\mathbf{C}$ are self-adjoint [5] but matrix $\mathbf{B}^H$ is not self-adjoint if the specific impedance Z has a complex component. Matrix $\mathbf{D}^H$ is calculated by swapping the left and right vectors of the outer product in Eq. (14d) and replacing the FTF with its conjugate, FTF*.

The matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ from Eq. (14) are easily expressed in UFL, and can readily be assembled into MPI distributed matrices compatible with PETSc using standard DOLFINx functionality. The details of creation of the matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are presented in Appendix A.1.

2.4.2 Typical boundary conditions

There are three typical boundary conditions in acoustics: Dirichlet, Neumann, and Robin, all of which can be expressed through Eq. (11):

1. For open boundaries (Dirichlet), $Z \to 0$ and $\hat{p}_1 = 0$.
2. For closed boundaries (Neumann), $Z \to \infty$ because $\hat{\mathbf{u}}_1$ is zero. So $\nabla\hat{p}_1 \cdot \mathbf{n} = 0$.
3. For other boundaries (Robin), Z is a finite complex number that quantifies acoustic radiation and phase shift at the boundary.

In helmholtz-x, Neumann boundaries are imposed naturally through the FEM discretization. For Dirichlet boundaries, degree of freedom (DOF) indices of the nodes on those surfaces are collected as a list. We use this to modify $\mathbf{A}$ and $\mathbf{C}$. For Robin and its special cases, choked inlet and choked outlet, we define the weak forms for these boundaries and use them to build $\mathbf{B}$. Z can be imposed on Robin boundaries through the reflection coefficients,[6] R, with Eq. (14b). The UFL implementation of the Robin boundary condition is shown in Appendix A.1.

[5] $\mathbf{A}^H = \mathbf{A}$ and $\mathbf{C}^H = \mathbf{C}$

[6] $Z = (1 + R)/(1 - R)$

In thermoacoustics, most of the facets are assumed to be Neumann or choked boundary conditions. The reflection coefficient of the inlet choked boundary condition is [3]

$$R_{in} = \frac{1 - \gamma_{in} M_{in}/(1 + (\gamma_{in} - 1)M_{in}^2)}{1 + \gamma_{in} M_{in}/(1 + (\gamma_{in} - 1)M_{in}^2)}, \tag{17}$$

where γ_{in} is the heat capacity ratio on the inlet choked boundary and M_{in} is the Mach number near the downstream of the inlet choked boundary. The UFL implementation of the choked inlet boundary condition is shown in Appendix A.1. Similarly, we write the choked outlet condition [3]

$$R_{out} = \frac{1 - (\gamma_{out} - 1)M_{out}/2}{1 + (\gamma_{out} - 1)M_{out}/2}, \tag{18}$$

where γ_{out} is the heat capacity ratio on the outlet choked boundary and M_{out} is the Mach number near the upstream of the outlet choked boundary. The UFL of the choked outlet boundary condition is implemented by changing R, γ and M in Appendix Appendix A.1.

2.5 Implementation of the flame matrix

The flame matrix, $\mathbf{D}$ contains an outer product between two sparse vectors, which requires careful implementation. The relation

$$(\gamma - 1)\frac{q_0}{u_b}\int_\Omega \phi_j h(\mathbf{x})\, \mathbf{dx} \int_\Omega \frac{w(\mathbf{x})}{\rho_0} \nabla\phi_k \cdot \mathbf{n}_r \mathbf{dx} \tag{19}$$

is shared between Eq. (14d) and (16d). For computational efficiency, we first perform the calculation of this cross product and compute the direct and adjoint submatrices, $\mathbf{D}_{ij}$ and $\mathbf{D}_{ji}$. Then we multiply the submatrices with FTF or (FTF)*, to obtain the direct or adjoint $\mathbf{D}$. The left and right components of Eq. (19) are calculated separately during assembly:

$$\underbrace{(\gamma - 1)\frac{q_0}{u_b}\int_\Omega \phi_j h(\mathbf{x})\, \mathbf{dx}}_{\text{left vector}} \underbrace{\int_\Omega \frac{w(\mathbf{x})}{\rho_0} \nabla\phi_k \cdot \mathbf{n}_r \mathbf{dx}}_{\text{right vector}} \tag{20}$$

The left and right vectors in Eq. (20) are swapped when generating the adjoint $\mathbf{D}$.

In helmholtz-x, two different flame matrices are implemented: one for a distributed measurement function $w(\mathbf{x})$ and the other for a pointwise measurement function $w(\mathbf{x}_r)$. The right vector of Eq. (20) is implemented differently for the pointwise flame matrix, as explained in Sect. 2.5.2. For the distributed flame matrix, Eq. (20) remains the same (Sect. 2.5.1). Appendix A.2 explains how parallel assembly is handled for the distributed and pointwise matrices.

2.5.1 Distributed measurement function

The measurement region $w(\mathbf{x})$ can take any shape. We choose a truncated Gaussian distribution that integrates to 1. This distribution introduces more nonzero contributions to the flame matrix, so is less sparse than the pointwise flame matrix.

In helmholtz-x, we use distributed $w(\mathbf{x})$ for longitudinal combustors and pointwise $w(\mathbf{x})$ for annular combustors. Although distributed $w(\mathbf{x})$ can be used for annular combustors, it has a high memory requirement.

2.5.2 Pointwise measurement function

The pointwise measurement function has a nonzero contribution only at the measurement point(s) ($\mathbf{x}_r$). We use this for annular combustors, where multiple pairs of $w(\mathbf{x}_r)$ and $h(\mathbf{x}_f)$ exist. We calculate the pointwise values of the gradient of the trial function $\nabla\phi_k$ near the points $\mathbf{x}_r$ such that Eq. (19) becomes

$$\underbrace{(\gamma - 1)\frac{q_{0_f}}{u_b}\int_\Omega \phi_j\, h(\mathbf{x})\, \mathbf{dx}_f}_{\text{left vector}} \underbrace{\int_\Omega \frac{\nabla\phi_k(\mathbf{x}_{r_f}) \cdot \mathbf{n}_r}{\rho_0(\mathbf{x}_{r_f})} \mathbf{dx}_f}_{\text{right vector}} \tag{21}$$

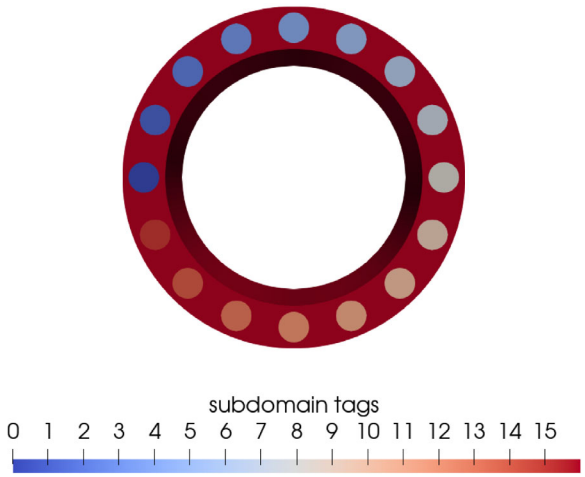
where subscript $_f$ represents the relevant flame index. If there are N discrete sectors in the annular combustor, there are N measurement points and heat release rate distributions. We find the contributions to $\mathbf{D}$ of the corresponding flame and its measurement point iteratively. We access the nonzero data through the subscript $_f$. In helmholtz-x, each individual $h(\mathbf{x})$ integrates to 1 over the domain.[7] In addition, the heat release rate volumes are tagged as separate subdomains starting from 0 to $N-1$. These tags are used in helmholtz-x during assembly (Fig. 1a).

2.6 Bloch boundary condition

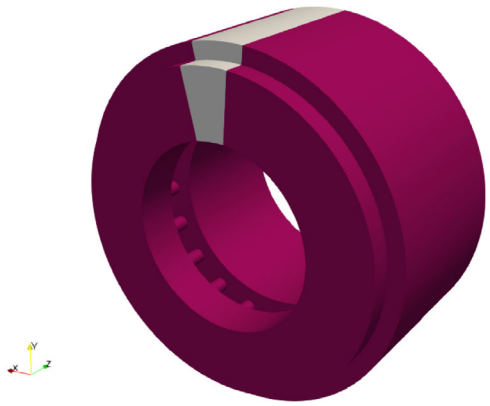
If the computational domain has an N-fold discrete rotational symmetry (Fig. 1b), the circumferential eigenmodes can be calculated by repeating a single geometry N times [42], first implemented in thermoacoustics by [5, 6]. For this, we apply Bloch-type boundary conditions to the relevant (master and slave) boundaries. This boundary condition reduces the computation load by 2^N times. helmholtz-x follows the methodology presented in [6]. According to Bloch-wave theory, the acoustic wave can be expressed by;

$$\hat{p}_b(\phi, r, z) = \hat{p}_{b+N}(\phi, r, z) e^{ib\phi}(\phi, r, z), \tag{22}$$

[7] The integral $\int_\Omega h(\mathbf{x})\mathbf{dx} = N$ over the domain and we input the heat release rate q_0 per single flame ($q_0 N = q_{total}$ where q_{total} is the total power of the annular combustor).



(a) Subdomain tagging for the annular combustor. The numbering runs from 0 to 15 because there are 16 flame volumes in total within the heat release rate function $h(\mathbf{x})$.



(b) Example annular combustor geometry. The gray section represents a single sector out of 20 identical sectors. With Bloch boundary condition, the azimuthal eigenmodes can be calculated by considering only the gray section.

Fig. 1 .

where ϕ, r and z are angular coordinates and b is the Bloch wavenumber. In helmholtz-x, the Bloch boundary condition is implemented as a Dirichlet (essential) boundary condition [6]. We map the matching nodes between master and slave boundaries (Fig. 2) with Eq. (22). When Bloch boundary conditions are applied, the number of DOFs in the mesh reduces. This requires manipulation of the matrices in Eq. (14) such that the DOFs of the slave boundary and its entries are deleted, and a periodicity scalar $f_b = e^{ib2\pi/N}$ is imposed on the master facets. The eigenmode of the system is found with these matrices. Then the slave DOFs are added back to the eigenvector.

In helmholtz-x, parallel calculations are handled by partitioning the mesh. This partition is performed arbitrarily. The parallelization of the calculations with Bloch boundary conditions is possible but is not addressed in helmholtz-x.

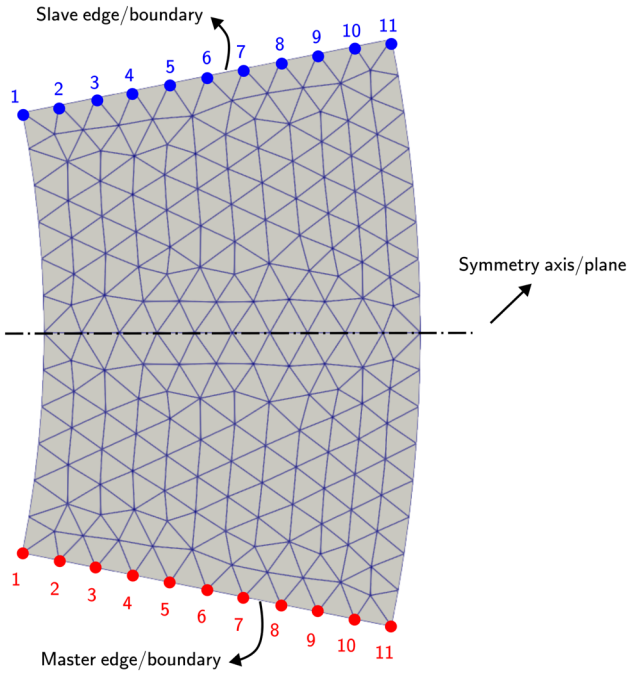


Fig. 2 Example mesh for Bloch BC application. The DOFs of the blue (slave) nodes should be paired with the DOFs of the red (master) nodes according to the numbers (from 1 to 11 in this example). The half-sector mesh is then reflected with respect to the symmetry axis/plane in order to guarantee one-to-one DOF mapping

This is because the pairing between master and slave DOFs is unevenly distributed over the processors, which makes the mapping difficult. There are several approaches that could handle this problem. One solution would be to allocate the DOFs of the master and slave nodes to certain processors via custom partitioning of the mesh. Another approach would be to use dolfinx_mpc[8] library to handle the parallelization through multi-point constraints. dolfinx_mpc currently does not, however, support complex coefficients as a periodic constraint, so this would have to be implemented in order to implement Bloch boundaries. The final approach would be to generate periodic meshes. We currently obtain periodicity by mirroring the mesh of the halved sector geometry with respect to the burner plane. A custom subroutine that imposes periodicity on the topology would need to be implemented to ensure that the mesh topology is periodic. These approaches require further development and we leave them for future work.

2.7 Fixed-point iteration & Newton's method

The nonlinear eigenvalue problem consists in finding the eigenvalues $\omega \in \mathbb{C}$ and the non-zero eigenvectors $\boldsymbol{p} \in \mathbb{C}^n$

[8] https://github.com/jorgensd/dolfinx_mpc.

such that

$$\boldsymbol{L}(\omega)\boldsymbol{p} = 0, \tag{23}$$

in which $\boldsymbol{L}$ depends nonlinearly on ω.

2.7.1 Fixed-point iteration

At each iteration, we solve a generalised eigenvalue problem for f^2.

$$\boldsymbol{L}(f;\omega)\boldsymbol{p} = \left(\boldsymbol{A} + \omega^{[k]}\boldsymbol{B} + f^2(\omega^{[k]})\boldsymbol{C} - \boldsymbol{D}(\omega^{[k]})\right)\boldsymbol{p} = 0, \tag{24}$$

where f is the eigenvalue and $\omega^{[k]}$ is a parameter. At the zeroth iteration, $\omega^{[k]} = 0$. In the simplest version of the fixed-point iteration, $\omega^{[k+1]} = f(\omega^{[k]})$, and the iteration might not converge to a fixed point. According to Banach's fixed-point theorem, a necessary condition for the mapping, f, to converge to a fixed point, ω, is that $|f'(\omega)| < 1$. In this case, there is no guarantee that $|f'(\omega)| < 1$. In order to improve the convergence, we can use a relaxation method:

$$\omega^{[k+1]} = g(\omega^{[k]};\alpha) \equiv \alpha f(\omega^{[k]}) + (1-\alpha)\omega^{[k]}, \tag{25}$$

where g is a mapping and α is a relaxation factor. The rate of convergence is equal to the smallest positive integer, m, satisfying $\partial^m/\partial z^m g(z=\omega) \neq 0$. If $g'(\omega) = 0$, then the rate of convergence is quadratic.

$$g'(\omega^{[k]};\alpha) = \alpha f'(\omega^{[k]}) + 1 - \alpha = 0. \tag{26}$$

From Eq. (26), we obtain $\alpha = 1/(1 - f'(\omega^{[k]}))$ and use it as a relaxation coefficient in Eq. (25). We can approximate $f'(\omega^{[k]})$ with a backward difference

$$f'(\omega) = \frac{f(\omega^{[k]}) - f(\omega^{[k-1]})}{\omega^{[k]} - \omega^{[k-1]}} \tag{27}$$

A fixed-point iteration can be implemented in more than one way. Instead of a linear eigenvalue problem, we can choose to solve at each iteration a quadratic eigenvalue problem for f [43];

$$\boldsymbol{L}(f;\omega)\boldsymbol{p} = \left(\boldsymbol{A} + f(\omega^{[k]})\boldsymbol{B} + f^2(\omega^{[k]})\boldsymbol{C} - \boldsymbol{D}(\omega^{[k]})\right)\boldsymbol{p} = 0. \tag{28}$$

We provide the pseudocode for the fixed point iteration scheme in Algorithm 1.

function `Eigensolver(`***A***, ***B***, ***C***, ***D***, *tol*, *maxiter*`)`**:**
  $k \leftarrow -1$
  $\omega^{[k]} \leftarrow 0$
  Solve $\boldsymbol{L}(f;\omega^{[k]})\boldsymbol{p} = 0$ to find f^2 and $\boldsymbol{p}$
  $\omega^{[k+1]} = f$
  $\Delta\omega \leftarrow 2 \times tol$
  while $|\Delta\omega| > tol$ **and** $k < maxiter$ **do**
    $k \leftarrow k + 1$
    Solve $\boldsymbol{L}(f;\omega^{[k]})\boldsymbol{p} = 0$ to find f^2 and $\boldsymbol{p}$
    $f' \leftarrow \dfrac{f(\omega^{[k]}) - f(\omega^{[k-1]})}{\omega^{[k]} - \omega^{[k-1]}}$
    $\alpha \leftarrow \dfrac{1}{1 - f'}$
    $\omega^{[k+1]} \leftarrow \alpha f + (1-\alpha)\omega^{[k]}$
    $\Delta\omega \leftarrow \omega^{[k+1]} - \omega^{[k]}$
  end
  return $\omega^{[k+1]}$, $\boldsymbol{p}$
end

Algorithm 1: Pseudocode for the fixed-point iteration algorithm. We supply the PETSc matrices **A**, **B**, **C** and **D**(ω) generated by DOLFINx and UFL to the algorithm. It converges to the eigenvalue by iteratively updating **D**(ω) when solving the polynomial eigenvalue problem in Eq. (28). Relaxation is used in order to accelerate the convergence.

2.7.2 Newton's method

At each iteration, we solve an auxiliary generalised eigenvalue problem for λ.

$$\boldsymbol{L}(\omega)\boldsymbol{p} = \lambda\boldsymbol{C}\boldsymbol{p}. \tag{29}$$

We want to find ω such that $\lambda = 0$. As with the fixed-point iteration, here ω takes the role of a parameter. If we linearize $\lambda = 0$ with respect to ω, we obtain

$$\lambda(\omega + \Delta\omega) \simeq \lambda(\omega) + \lambda'(\omega)\Delta\omega = 0. \tag{30}$$

Therefore, at each iteration step we update ω according to

$$\omega^{[k+1]} = g(\omega^{[k]}) \equiv \omega^{[k]} - \frac{\lambda(\omega^{[k]})}{\lambda'(\omega^{[k]})}, \tag{31}$$

where g is a mapping and g' is guaranteed to be 0 at a fixed point. Using perturbation theory, the first derivative of λ with respect to ω is

$$\lambda'(\omega) = \frac{\boldsymbol{p}^{\dagger H}\boldsymbol{L}'(\omega)\boldsymbol{p}}{\boldsymbol{p}^{\dagger H}\boldsymbol{C}\boldsymbol{p}} \tag{32}$$

For a degenerate eigenvalue with multiplicity m, in order to have quadratic convergence,

$$\omega^{[k+1]} = g(\omega^{[k]}) \equiv \omega^{[k]} - m\frac{\lambda(\omega^{[k]})}{\lambda'(\omega^{[k]})} \tag{33}$$

Newton's method is the first of a class of methods called Householder's methods [6, 10]. The pseudocode for the proposed algorithm with Newton's method is given in Alg. 2.

function `Eigensolver`($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\omega^{[0]}$, *tol*, *maxiter*)**:**
  $k \leftarrow -1$
  $\Delta\omega \leftarrow 2 \times tol$
  while $|\Delta\omega| > tol$ **and** $k < maxiter$ **do**
    $k \leftarrow k + 1$
    Solve $\boldsymbol{L}(\omega^{[k]})\boldsymbol{p} = \lambda \boldsymbol{C}\boldsymbol{p}$ to find λ and $\boldsymbol{p}$
    Solve $(\boldsymbol{L}(\omega^{[k]}))^H \boldsymbol{p}^\dagger = \lambda^* \boldsymbol{C}\boldsymbol{p}^\dagger$ to find λ^* and $\boldsymbol{p}^\dagger$
    $\lambda' \leftarrow \dfrac{\boldsymbol{p}^{\dagger H}\boldsymbol{L}'(\omega^{[k]})\boldsymbol{p}}{\boldsymbol{p}^{\dagger H}\boldsymbol{C}\boldsymbol{p}}$
    $\omega^{[k+1]} \leftarrow \omega^{[k]} - \dfrac{\lambda}{\lambda'}$
    $\Delta\omega \leftarrow \omega^{[k+1]} - \omega^{[k]}$
  end
  return $\omega^{[k+1]}$, $\boldsymbol{p}$, $\boldsymbol{p}^\dagger$
end

Algorithm 2: Pseudocode for the Newton's method. We supply the PETSc matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}(\omega)$ as well as initial guess for the eigenvalue $\omega^{[0]}$. We iteratively update ω with Newton's method using direct and adjoint eigenfunctions of Eq. (29). Unlike fixed-point iteration, this algorithm strongly relies on the initial eigenvalue $\omega^{[0]}$.

2.8 Software structure

helmholtz-x follows the philosophies of scalability, reproducibility, evolvability and aims to be readable using a Python interface [44]. helmholtz-x heavily exploits the principles of object oriented programming. The typical simulation flow is visualized in Fig. 3, in which the submodules of helmholtz-x and their functionalities are classified. The source code of helmholtz-x can be found in the `helmholtz_x` directory in the repository.[9] helmholtz-x uses DOLFINx v0.9.0.

2.8.1 Pipeline of helmholtz-x

In this section, we describe the helmholtz-x utilities step by step.

Mesh, subdomains, and facets: We first need to generate the mesh, subdomains and facets. If flames are included, we need to define the flame volume subdomains during mesh generation. These subdomains are labeled from 0 to $N-1$. We also tag facets to impose boundary conditions. helmholtz-x provides simple geometries such as intervals, squares and boxes using the DOLFINx mesh generators. For complex geometries, we use the open source finite element mesh generator Gmsh, which generate grids for .step files through its Python API and Open Cascade kernel [45]. For Gmsh meshes, we transform the generated grids into the XDMF format for consistency with DOLFINx modules. The following lines read an XDMF mesh with its subdomains and facets tags:

```
geometry = XDMFReader("PathForMesh")
mesh, subdomains, facet_tags = geometry
    .getAll()
```

There are several examples with different grids in the */numerical_examples* folder.

Assembling the acoustic matrices: We define the parameters for acoustic matrices **A**, **B** and **C** with a standalone *params.py* file that is imported into the main calculation file. These two files are kept separate in order to track the problem parameters more conveniently. First we define boundary conditions by specifying facet tags as a Python dictionary, for example:

```
boundary_conditions = {1:{"Dirichlet"
    },
                       3:{"
    ChokedInlet":params.M_in},
                       8:{"
    ChokedOutlet":params.M_out},
                       11:{"Robin":
    params.R}}
```

where the integer dictionary keys represent the corresponding Gmsh tags of each boundary condition. The choked inlet and choked outlet boundaries adopt the Mach number near the boundaries. Robin boundaries are specified with their reflection coefficients. We input the speed of sound or temperature field to construct the acoustic matrices:

```
c = params.c(mesh)
matrices = AcousticMatrices(mesh,
    facet_tags, boundary_conditions,
    c, degree=degree)
```

where the *degree* represent the polynomial degree of basis functions of the continuous Lagrange finite elements. The parameter c is the speed of sound. The *AcousticMatrices* class can also take temperature as a parameter and convert it to the speed of sound using the relation: $c = \sqrt{\gamma r_{gas} T_0}$.

Defining the flame transfer function: If we solve the inhomogeneous Helmholtz equation, matrix **D** needs to be implemented, which requires an FTF. helmholtz-x has two different FTFs: the $n-\tau$ formulation or the state space representation (from an experimental FTF). These can be defined by using *nTau* or *stateSpace* classes as

```
FTF = nTau(params.n, params.tau)
FTF = stateSpace(params.S1, params.s2
    , params.s3, params.s4)
```

by importing the necessary parameters from *params.py*.

[9] https://github.com/ekremekc/helmholtz-x/tree/paper.

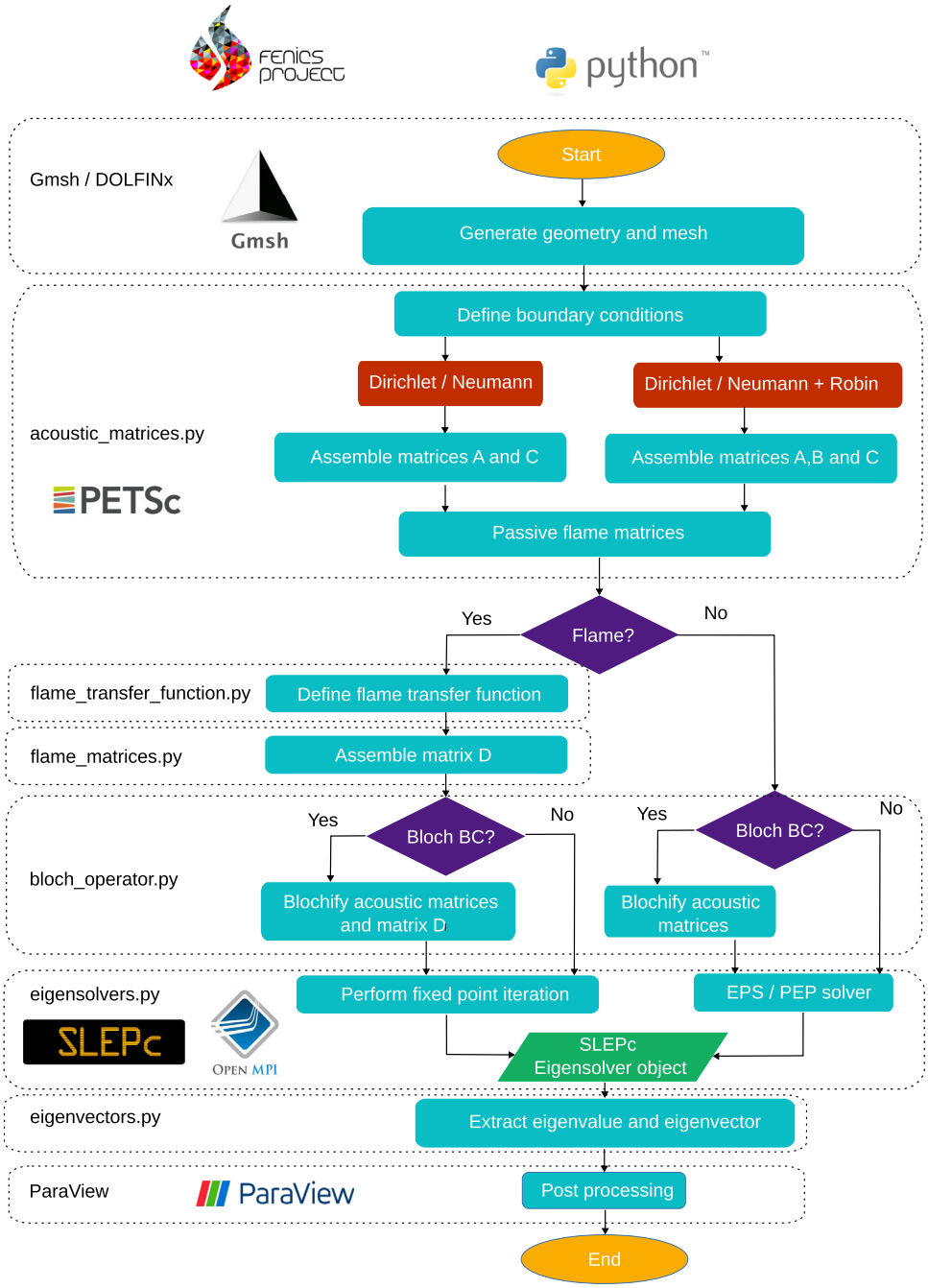


Fig. 3 The components of helmholtz-x and the flowchart for the solution of the inhomogeneous thermoacoustic Helmholtz equation

Assembling the flame matrix: For distributed **D**, we define the parameters of Eq. (19) and input them to the *DistributedFlameMatrix* class with:

```
rho = rho_step(mesh, params.x_f,
    params.a_f, params.rho_d, params.
    rho_u)
w = gaussianFunction(mesh, params.x_r
    , params.a_r)
h = gaussianFunction(mesh, params.x_f
    , params.a_f)
FTF = nTau(params.n, params.tau)
D = DistributedFlameMatrix(mesh, w, h
    , rho, T, params.q_0, params.u_b,
     FTF, degree=degree)
D.assemble_submatrices()
```

where the function *D.assemble_submatrices()* takes two parameters, 'direct' (by default) or 'adjoint'. For implementing pointwise **D**, we import the necessary parameters of Eq. (21) and use them in the *PointwiseFlameMatrix* class by, for example:

```
h = Q_multiple(mesh, subdomains,
    params.N_sector)
D = PointwiseFlameMatrix(mesh,
    subdomains, params.x_r, h, params
    .rho_xr, params.q_0, params.u_b,
    FTF, degree=degree)
D.assemble_submatrices()
```

Imposing Bloch boundary conditions: If there are Bloch boundaries, we define them in the *boundary_conditions* dictionary. For this, we specify slave and master boundaries with their physical tags such as

```
boundary_conditions = {11: {'Robin':
    params.R_outlet},
                       12: 'Master',
                       13: 'Slave'}
```

Then we manipulate the matrices in the system with

```
bloch_matrices = Blochifier(geometry,
     boundary_conditions, N,
    acoustic_matrices)
D = PointwiseFlameMatrix(mesh,
    subdomains, params.x_r, h, params
    .rho_amb, params.q_0, params.u_b,
     FTF, degree=degree, bloch_object
    =bloch_matrices)
D.blochify()
```

where N is the Bloch number. The flame matrix classes take the post-Bloch matrices as a bloch_object parameter to create **D**.

Solving the system: If the Helmholtz equation is homogeneous, we use the EPS solver such that

```
target = 200 * 2 * np.pi
E = eps_solver(matrices.A, matrices.C
    , target, nev=2, print_results=
    True)
```

or, if we have Robin boundaries, the PEP solver such that

```
target_dir = 262 * 2 * np.pi
E = pep_solver(matrices.A, matrices.B
    , matrices.C, target_dir, nev=10,
     print_results=True)
```

In helmholtz-x, the unit of the target eigenvalue is rad s^{-1}. We converge to the targeted angular eigenfrequency. If the problem is inhomogeneous, we have **D** and we use fixed point iteration (or a Newton solver) such that

```
target = 200 * 2 * np.pi
E = fixed_point_iteration(matrices, D
    , target, nev=2, i=0,
    print_results= False)
```

All these functions return a SLEPc object E, from which we extract eigenvalues and eigenvectors.

Extracting the eigenvalues and eigenvectors: In helmholtz-x, we normalize the eigenvectors such that $\int \hat{p}^2_{1,h}$=1. When the object E is computed, it has eigenvalue ω and eigenvectors $\mathbf{p}$ and $\mathbf{p}^\dagger$ as instances. We extract them with the *normalize_eigenvector* function with

```
omega, p = normalize_eigenvector(mesh
    , E, i=0, degree=degree, which='
    right')
```

where the parameter i is the index of the converged eigenvalue and the keyword *which* decides whether to return the right or left eigenvector.

Saving the eigenvector and eigenvalue: We save the eigenvector in XDMF file format with utility *xdmf_writer*, if the degree of the finite element basis is 1. For the higher order Lagrange spaces with the degree greater than 1, we use VTK file format by calling *vtk_writer*. We save the eigenvalue of the corresponding eigenvector with *dict_writer* function.

```
# Save eigenvectors
xdmf_writer("PathToWrite", mesh, p) #
     if degree of p is 1
vtk_writer("PathToWrite", mesh, p) #
    if degree of p is greater than 1
# Save eigenvalues
omega_dict = {'direct':omega}
dict_writer("PathToWrite", omega_dict
    )
```

We then visualize the resulting output file with open-source visualization toolkit ParaView [46].

2.8.2 Parallelization

Any eigenmode calculation without Bloch boundary conditions can be parallelized using helmholtz-x with the command *mpirun -np n_proc python3 -u file.py* where *n_proc* specifies the number of processors and *file.py* is the Python script to be parallelized. After writing the *params.py* and main scripts following the pipeline (Sect. 2.8.1), helmholtz-x handles the parallelization internally. All computations in this paper are performed using hardware with Intel(R) Xeon(R) E5-2620 v4 2.10 GHz x 16 processors and 32 GB memory.

3 Numerical examples

In this section, we present several test cases with helmholtz-x for longitudinal and annular geometries and compare them

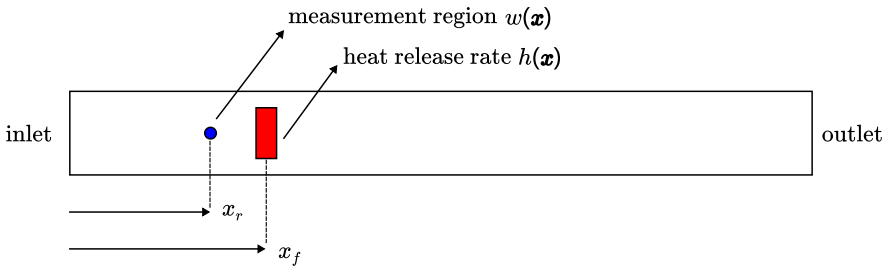


Fig. 4 Schematic representation of the Rijke tube. We implement $w(\mathbf{x})$ and $h(\mathbf{x})$ with Gaussian functions

against other numerical tools in the literature. An additional numerical verification against manufactured solution without the thermoacoustic effect is given in Appendix B.1.

3.1 Longitudinal thermoacoustic systems

We first verify the results of helmholtz-x in relatively simple thermoacoustic cases: a hot wire Rijke tube, a longitudinal combustor, and an industrial network model [2].

3.1.1 Hot wire Rijke tube

In this section, 1D, 2D and 3D test cases of the hot wire Rijke tube are implemented. The code is presented in `numerical_examples / Longitudinal / NetworkCode/RijkeTube*` folders in the repository. The schematic representation of the hot wire Rijke tube case is shown in Fig. 4.

We define $w(\mathbf{x})$ and $h(\mathbf{x})$ as multi-dimensional Gaussian functions (Eq. 34) in which n_{dim} is the spatial dimension and σ controls the width of the Gaussian around its central point $P(x_0, y_0, z_0)$:

$$G(\mathbf{x}) = \frac{1}{\sigma^{n_{dim}}(2\pi)^{n_{dim}/2}} \exp\left(-\frac{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 + \ldots}{2\sigma^2}\right). \tag{34}$$

We supply $w(\mathbf{x})$ and $h(\mathbf{x})$ fields as inputs when constructing the *DistributedFlameMatrix* instance for assembling the flame matrix. The parameters of the Helmholtz solver are tabulated in Table 1.

The speed of sound field is calculated from the temperature distribution. The interaction index n is scaled by dividing by the cross-sectional area of the tube ($\pi d^2/4$) for the 1D case and by $\pi d/4$ for the 2D case. The 3D case does not require scaling. For calculation of $\gamma = c_p/c_v = c_p/(c_p - r_{gas})$, linear temperature dependence of $c_p(T) = 973.60091 + 0.1333T$ is used for both the Helmholtz solver and the network code. For simplicity, all boundaries are assumed to be Neumann. Passive and active flame simulations are performed. Their eigenfrequencies are tabulated in Table 2 and they agree well. The adjoint eigenmode computation with helmholtz-x is explained with a more detailed test cases in Appendix B.2.

Table 1 Dimensional parameters of the hot wire Rijke tube

Parameter	Value	Unit
L	1	m
d	0.047	m
r_{gas}	287.1	J kg^{-1}K^{-1}
p_0	101325	Pa
ρ_u	1.22	kg m^{-3}
ρ_d	0.85	kg m^{-3}
T_u	285.6	K
T_d	409.92	K
q_0	-27.0089	W
u_b	0.1006	m s^{-1}
n	0.1	–
τ	0.0015	s
x_f	0.25	m
a_f	0.025	–
x_r	0.2	m
a_r	0.025	–

3.1.2 Flame in a cylindrical duct with choked boundaries

In this case, we use a geometry that has area changes and choked boundaries at both ends. The code is implemented in `numerical_examples/Longitudinal/NetworkCode/FlamedDuct` folder in the repository. A schematic representation of this test case is shown in Fig. 5.

This example is useful to check the Helmholtz solver's ability to capture the influence of the area change and acoustic energy losses through the choked boundaries. The parameters of this test are given in Table 3.

The density field ρ_0 is calculated from the ideal gas equation of state, $p_0 = \rho_0 r_{gas} T_0$ using the temperature field. The Mach number near the inlet is $M_{in} = 0.0092$ and near the outlet is $M_{out} = 0.011$. The heat release rate and measurement region fields for this case are implemented with Eq.

Table 2 Eigenfrequencies of the passive and active flame test cases for the Rijke tube

Run	Passive		Active	
	f (1/s)	GR (rad/s)	f (1/s)	GR (rad/s)
Network code [2]	169.178074	0	197.784121	6.411332
1D helmholtz-x	169.377645	0	197.699903	6.683160
2D helmholtz-x	169.377337	0	197.762459	6.668631
3D helmholtz-x	169.410068	0	198.577709	6.797977

GR denotes the growth rate. The eigenfrequencies become closer as the grid resolutions of helmholtz-x increases

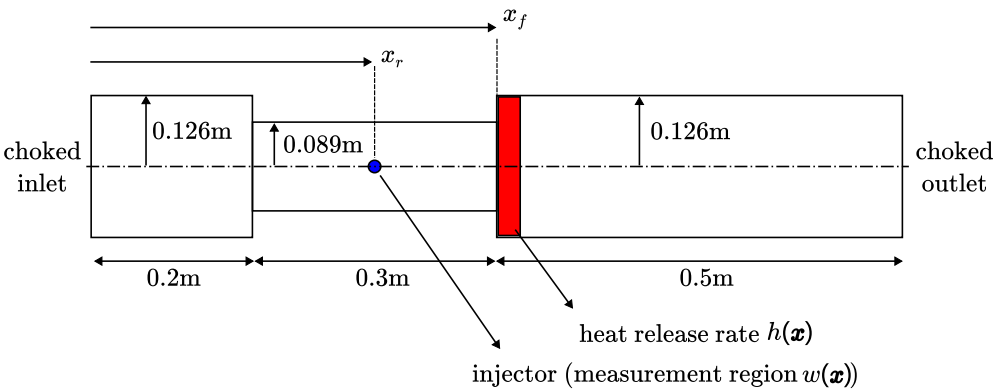


Fig. 5 Schematic representation of the flame in a cylindrical duct with choked boundary conditions. The red zone represents the heat release rate field and the blue zone shows the fuel injection point, which is at the centre of the 0.3 m duct

Table 3 Dimensional parameters of the flame in a cylindrical duct

Parameter	Value	Unit
r_{gas}	287.1	J kg^{-1}K^{-1}
p_0	101,325	Pa
T_u	1000	K
T_d	1500	K
q_0	−57015.232	W
u_b	11.4854	m s^{-1}
n	1	–
τ	0.002	s
x_f	0.5	m
a_f	0.025	–
x_r	0.35	m
a_r	0.025	–

T_u denotes the temperature before the flame and T_d denotes the temperature after the flame. γ linearly depends on the temperature as in Sect. 3.1.1

(34). For the heat release rate field, the Gaussian function is halved and rescaled such that it integrates to 1.

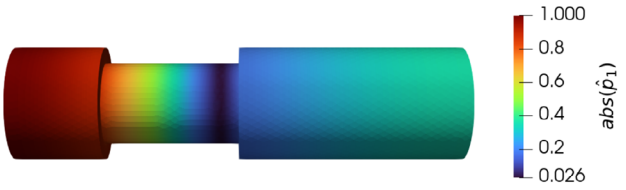
Table 4 shows the eigenvalues for passive and active flame configurations, comparing helmholtz-x against the network code. For the passive flame, the thermoacoustic system loses energy through the choked boundaries as expected. For the active flame, the growth rate becomes more negative.

The normalized magnitudes of the eigenfunctions of helmholtz-x and a network model for the active flame case can be seen in Fig. 6. The trend of the acoustic pressure in the axial direction is very similar for both approaches. We note that the network model for this thermoacoustic case includes some connections, each representing the thermoacoustic contributions such as choked ends, flame and area change affects. Although the absolute acoustic pressure for the network code looks continuous (Fig. 6b), the network code performs pointwise evaluations and performs linear interpolation between them.

Table 4 Eigenfrequencies of the passive and active flame test cases for the flame in a duct

Run	Passive frequency (1/s)	GR (rad/s)	Active frequency (1/s)	GR (rad/s)
Network model [2]	267.1030	−10.944425	267.307657	−43.4478
helmholtz-x	261.7945	−11.9214	262.559781	−43.2349

GR denotes the growth rate. 177,737 elements are used for the FEM simulation



(a) Helmholtz solver

(b) Network model

Fig. 6 Normalized amplitude of the direct eigenfunction $\hat{p}_1$ for **a** the Helmholtz solver and **b** a network code

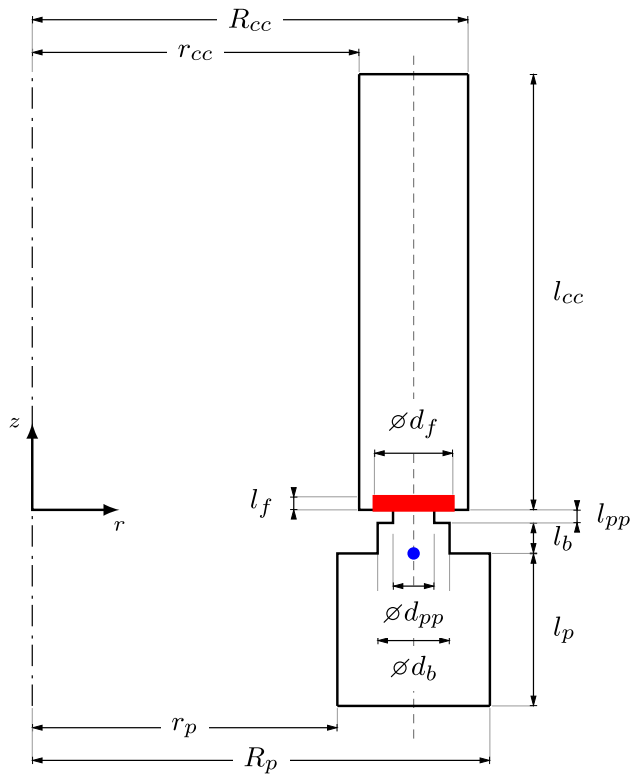


Fig. 7 Section of one sector of the MICCA combustor. The dash-dotted line is the axis of symmetry. The subscripts stand for: plenum (p), burner (b), perforated plate (pp), flame (f), combustion chamber (cc). $r_p = 140$ mm, $R_p = 210$ mm, $l_p = 70$ mm, $d_b = 33$ mm, $l_b = 14$ mm, $d_{pp} = 18.9$ mm, $l_{pp} = 6$ mm, $d_f = 36$ mm, $l_f = 6$ mm, $r_{cc} = 150$ mm, $R_{cc} = 200$ mm, $l_{cc} = 200$ mm. The vertical dashed axis represents the longitudinal axis of the burner. The red zone represents the cylindrical heat release rate domain and the blue circle represents the pointwise measurement function

3.2 Annular combustors

In this section, we demonstrate the capability of helmholtz-x to compute thermoacoustic eigenmodes for annular geometries. The helmholtz-x code is held in the `numerical_examples/AnnularCombustor/MICCA` folder in the repository. For this test case, we choose a laboratory-scale annular combustor, MICCA [47, 48]. Thermoacoustic limit cycles of standing, spinning, and slanting modes are observed at some operating conditions [49, 50]. The MICCA combustor is composed of an annular plenum, 16 injectors and an annular combustion chamber. Each injector has a burner and a perforated plate. Following [48], the perforated plate and the burner are represented by a cylindrical volume. Figure 7 shows one sector of the MICCA combustor model.

For annular geometries, we use the *PointwiseFlameMatrix* class to implement **D** (see Sect. 2.5.2). We consider the same operating conditions as operating point B in [48]. A standing mode with a stable limit cycle at a frequency of 487 Hz is observed in the experiments. The total power of the flame for each burner is $q_0 = 2080$ W, and the bulk flow velocity is $u_b = 0.66$ m/s. The ratio of specific heats, $\gamma = 1.4$, is assumed to be independent of temperature. The mean temperature in the plenum and up to the combustion chamber is $\bar{T} = 300$ K. In the combustion chamber, the temperature profile is parabolic, gradually decreasing between the values at the flame positions $\mathbf{x}_f$ and the chamber outlet, given in Eq. (35). In helmholtz-x, Eq. (35) is implemented with degree 0 discontinuous Galerkin elements.

$$T(z) = \begin{cases} 300, & \text{if } z < z_f \\ (1200 - 1521)\left(\frac{z - z_f}{l_{cc}}\right)^2 + 1521, & \text{otherwise.} \end{cases} \tag{35}$$

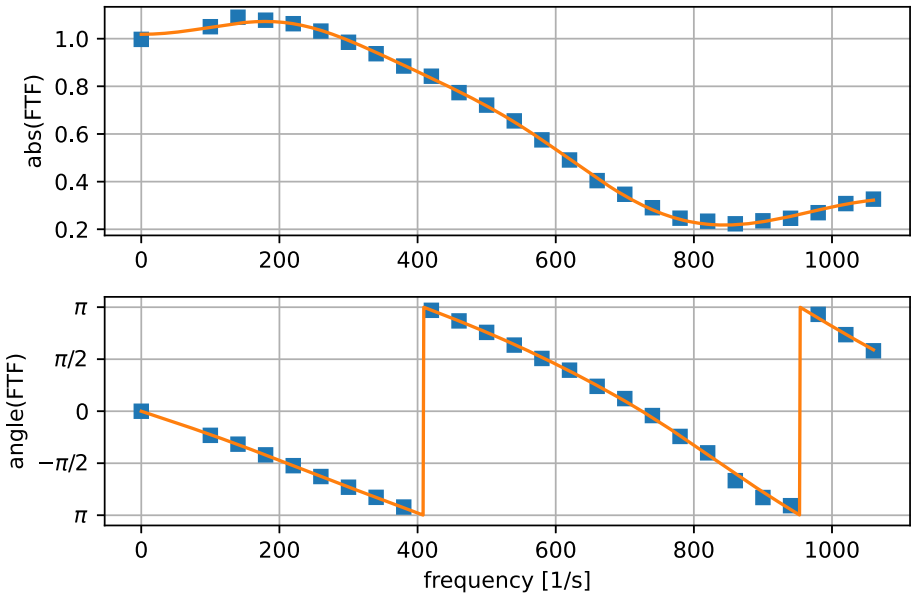


Fig. 8 Gain and phase of the flame transfer function ($|u'/\bar{u}| = 0.1$) as a function of the frequency. The squares are the values obtained from the experiments [48], and the solid line is the transfer function of the linear state-space model, evaluated at real values of the angular frequency ω. The *stateSpace* class in helmholtz-x is used to obtain an analytical function for FTF(ω)

The experimental flame transfer function depends on the frequency of the excitation and on the ratio of the root mean square of the velocity fluctuation measured at the reference point, u_1, to the average flow velocity in the injector, u_b (Eq. 3).

We apply Neumann boundary conditions at the combustor walls and a Robin boundary condition at the outlet surface. The reflection coefficient at the outlet boundary is $R_{outlet} = -0.875 - 0.2i$. In this paper, we obtain the flame transfer function FTF, by considering a relatively small amplitude, $|u'/\bar{u}| = 0.1$. In order to calculate the first derivative of the linear operator $\mathcal{L}$ with respect to the eigenvalue ω without approximations, we need FTF(ω) in Eq. (3) to be analytic in the complex plane [51]. We approximate the frequency response of the flame with a linear state-space model. The transfer function of the state-space model,

$$\text{FTF}(\omega) = \mathbf{s}_3^T \left(i\omega\mathbf{I} - \mathbf{S}_1\right)^{-1}\mathbf{s}_2 + s_4, \tag{36}$$

will correspond to the FTF. In order to obtain an analytic transfer function, we apply the Vector Fitting algorithm [51, 52]. The experimental FTF and the transfer function of the state-space model are shown in Fig. 8.

3.2.1 Eigenmodes

helmholtz-x can capture numerous eigenmodes by specifying the nearest target to the desired eigenvalue. Computations for different eigenfunctions are shown in Fig. 9 and their eigenfrequencies are presented in Table 5.

3.2.2 Bloch boundary conditions

In this section, we check the Bloch boundary condition implementation within helmholtz-x. We use a single sector of MICCA and the same parameters as in Sect. 3.2. As explained in Sect. 2.8.1, we impose slave and master boundaries of Bloch boundaries with their physical tags. We calculate the Bloch form of the acoustic and flame matrices for MICCA. For this comparison, we only consider the plenum-dominant azimuthal mode (Fig. 9c) and we verify the results against [5]. The results for different case studies, including with parallelization, are tabulated in Table 6.

The experimental eigenfrequency found in [48] is observed to be lower than that calculated with helmholtz-x and PyHoltz [6]. This may be due to the approximated speed of sound field (Eq. 35) used in both numerical computations. helmholtz-x has slightly different eigenvalues for the full annulus and the case with Bloch boundary conditions. This is because periodicity is not imposed when generating the full annulus mesh. Computations with the full annulus mesh generated by copying and rotating the single sector mesh N_{sector} times would give closer eigenvalues, as performed in [6]. This requires further manipulation of the physical tags when using Gmsh so we do not address this here. Although it uses fewer elements, PyHoltz [6] has a higher computational time than helmholtz-x because it uses the numpy and scipy packages for matrix generation and solution, rather than PETSC and SLEPc [53]. When parallelizing from 1 to 8 processors, the computational time of helmholtz-x reduces by a factor of 8. For similar test cases, SLEPc shows linear scaling up to 16 processors (Sect. 9 in [54]).

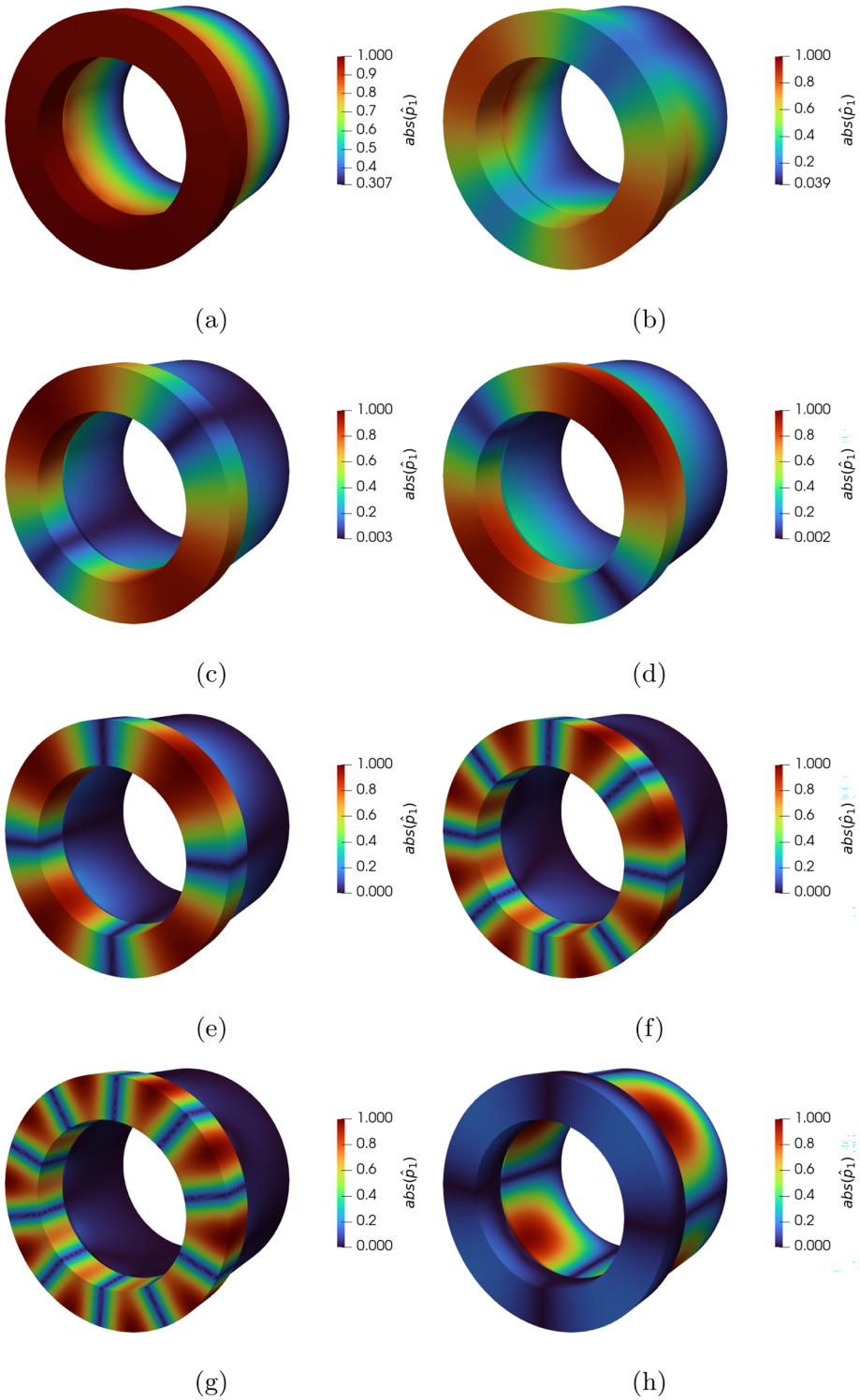


Fig. 9 Computed eigenmodes for the MICCA combustor with helmholtz-x. The corresponding eigenvalues are given in Table 5

Table 5 Eigenfrequencies of the active flame test cases for the MICCA combustor using helmholtz-x

Mode	Active frequency (1/s)	GR (rad/s)
Fig. 9a	149.151	−534.155
Fig. 9b	289.976	−629.029
Fig. 9c	517.364	+465.643
Fig. 9d	517.355	+435.378
Fig. 9e	721.206	+3.871
Fig. 9f	1314.411	−5.202
Fig. 9g	1617.749	−22.147
Fig. 9h	1721.129	+333.431

The corresponding modeshapes are shown in Fig. 9. The growth rates of Fig. 9c, d become closer when the numerical grid is refined

Table 6 Eigenfrequencies of the active flame test cases for the MICCA combustor calculated with fixed point iteration

Case	Tool	Number of processors	Number of cells	Eigenfrequency (1/s)	Computation time (s)
Experiment	[48]	–	–	487	–
Full annulus	[6]	1	10,528	511.4+79.4j	4627.55
Bloch	[6]	1	658	511.4+79.4j	82.88
Full annulus	helmholtz-x	1	163,165	517.3+74.1j	122.47
Full annulus	helmholtz-x	8	167,401	517.3+74.1j	14.01
Bloch	helmholtz-x	1	47,672	513.3+75.6j	15.70

4 Adjoint based shape optimization

In this section, we demonstrate the adjoint features of helmholtz-x and apply them to shape optimization. We consider a hot wire Rijke tube, as in Appendix B.1, with dimensional equations. We parametrize the cylindrical geometry using FFD. The helmholtz-x code for this example is in the `numerical_examples/ShapeSensitivities/RijkeFFD` folder in the repository.

4.1 Free form parametrization

Free form deformation establishes a parametrization relationship between the mesh nodes and the individual control points around and inside the geometry of interest. These control points generate the control lattice (Fig. 10a), which can form any geometric shape. Mostly, cylindrical or cube-shaped lattices are preferred so that the control points can be manipulated conveniently.

Any mesh node around the control lattice can be expressed in terms of parametric coordinates (s, t, u), as in Eq. (37), where $\mathbf{X}_0$ represents the center of the control lattice and $\mathbf{S}$, $\mathbf{T}$ and $\mathbf{U}$ are the parametric unit vectors in the radial, circumferential (azimuthal) and axial directions, respectively.

$$\mathbf{X} = \mathbf{X}_0 + s\mathbf{S} + t\mathbf{T} + u\mathbf{U}. \tag{37}$$

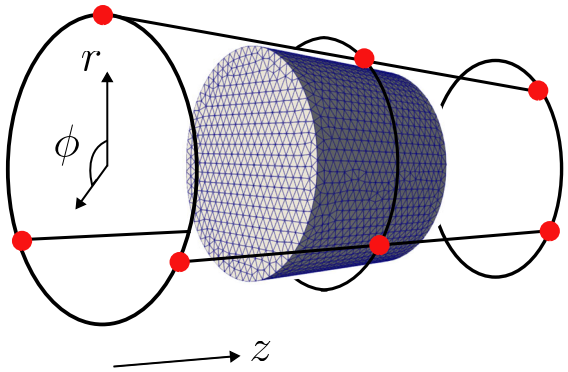
Considering the control lattice in Fig. 10a, the mesh nodes are firstly transformed into cylindrical coordinates. Then their parametric coordinates are computed with Eq. (37). The range of the parametric coordinates is between 0 and 1 for radial (r) and axial (z) directions and between 0 and 2π for the azimuthal direction (ϕ).

The FFD control points can be arbitrarily inserted depending on the application. In this paper, we specify the positions of the control points with a equispaced pattern forming a cylindrical lattice using Eq. (38). We define the FFD control points using

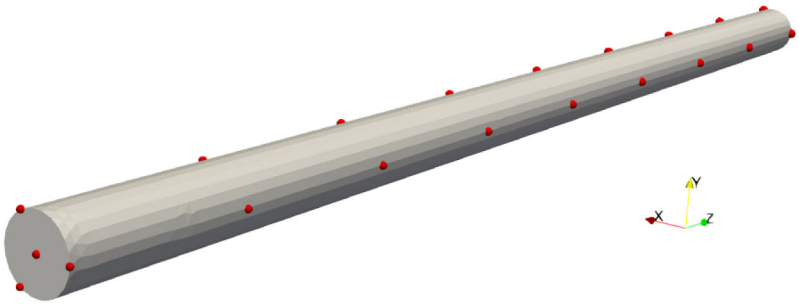
$$\mathbf{P}_{ijk} = \mathbf{X}_0 + \frac{i}{l}\mathbf{S} + \frac{j}{m}\mathbf{T} + \frac{k}{n}\mathbf{U}, \tag{38}$$

where l, m, n specifies the total number of control points in the radial, azimuthal and axial directions. The positions and number of FFD control points are important because they form the control lattice and determine the permitted deformation magnitudes and directions. Therefore, the control points should be numbered and positioned to prevent potential overlapping geometrical deformations after control point displacements. For simple or symmetric geometries, equispaced control points might handle the deformations. However, for complicated geometries, an irregular pattern for the placements might perform better depending on the goal. For instance, control points structuring a cylindrical lattice could handle cylinder-like geometries better, whereas

Fig. 10 .



(a) FFD configuration for a cylindrical grid using a cylindrical control lattice with control points (red dots). r, ϕ and z denotes the radial, circumferential and axial directions. Black lines connect the control points.



(b) FFD control points for the hot wire Rijke tube. There are 2, 4 and 9 control points positioned through the radial, circumferential and axial directions respectively.

for cornered geometries, box-like lattices might be more convenient, with control points positioned at the corners.

We first position the FFD control points within the lattice and calculate the parametric coordinates of the mesh nodes. We change the coordinates of the FFD control points and deform the mesh nodes individually with trivariate Bernstein basis polynomials, as shown in Eq. (39):

$$\mathbf{X}_{FFD} = \left(\sum_{i=0}^{l}\binom{l}{i}(1-s)^{l-i}s^{i}\left(\sum_{j=0}^{m}\binom{m}{j}(1-t)^{m-j}t^{j}\left(\sum_{k=0}^{n}\binom{n}{k}(1-u)^{n-k}u^{k}\mathbf{P}_{ijk}\right)\right)\right) \tag{39}$$

The FFD configuration for the Rijke tube can be seen in Fig. 10b. We place more control points in the axial direction than other directions in order to increase the longitudinal control.

4.2 Shape derivatives

The shape sensitivities for the thermoacoustic Helmholtz equation are derived in [39]. In Hadamard-form, we can compute the shape derivative of the FFD control points using direct and adjoint eigenvectors. The most general expression for the shape derivative is that using impedance (Robin) boundary conditions, as in Eq. (40):

$$\omega'_{ijk} = \int_{\partial\Omega} \mathbf{V}_{ijk}\cdot\mathbf{n}_{ijk}\Big(-\hat{p}_1^{\dagger *}\Big(\kappa c^2\frac{\partial c}{\partial n}\Big)\frac{\partial \hat{p}_1}{\partial n} + \nabla\cdot\Big(\hat{p}_1^{\dagger *}c^2\nabla\hat{p}_1\Big) - 2\frac{\partial \hat{p}_1^{\dagger *}}{\partial n}c^2\frac{\partial \hat{p}_1}{\partial n}\Big)dS, \tag{40}$$

where ω'_{ijk} is the complex-numbered shape derivative for the control point $\mathbf{P}_{ijk}$ and $\mathbf{n}_{ijk}$ is its outward normal vector. When applying Neumann boundaries, we impose $\partial\hat{p}_1/\partial n = 0$ and $\partial\hat{p}_1^{\dagger}/\partial n = 0$. In this example, we only consider design changes for the lateral surface in the normal directions. The shape derivative of any control point is then

$$\omega'_{ijk} = \int_{\partial\Omega} \mathbf{V}_{ijk}\cdot\mathbf{n}_{ijk}\Big(\nabla\cdot\Big(\hat{p}_1^{\dagger *}c^2\nabla\hat{p}_1\Big)\Big)dS, \tag{41}$$

for Neumann boundary conditions. We use Eq. (41) to calculate the shape derivatives of the control points visualized in Fig. 10a in the direction of facet normals. To compute the displacement field $\mathbf{V}_{ijk}$ for the control point $\mathbf{P}_{ijk}$, we take the derivative of the mesh nodes with respect to the control point, as shown in Eq. (42).

$$\frac{\partial}{\partial \mathbf{P}_{FFD}}\left(\mathbf{X}_{ffd}\right) = \mathbf{V}_{ijk} = \Big(\sum_{i=0}^{l}\binom{l}{i}(1-s)^{l-i}s^{i}\Big(\sum_{j=0}^{m}\binom{m}{j}(1-t)^{m-j}t^{j} \Big(\sum_{k=0}^{n}\binom{n}{k}(1-u)^{n-k}u^{k}\Big)\Big)\Big). \tag{42}$$

The field $\mathbf{V}_{ijk}$ can then be used in Eq. (41) to calculate ω'_{ijk} for $\mathbf{P}_{ijk}$. The helmholtz-x implementation of Eq. (41) is shown in Listing 1.

```
1  def shapeDerivativesFFD(geometry,
       lattice, physical_facet_tag,
       omega_dir, p_dir, p_adj, c,
       acousticMatrices, FlameMatrix):
2      normal = FacetNormal(geometry.
       mesh)
3      ds = Measure('ds', domain =
       geometry.mesh, subdomain_data =
       geometry.facet_tags)
4      p_adj_norm = normalize_adjoint(
       omega_dir, p_dir, p_adj,
       acousticMatrices, FlameMatrix)
5      p_adj_conj = conjugate_function(
       p_adj_norm)
6      G_neu = div(p_adj_conj * c**2 *
       grad(p_dir))
7      derivatives = {}
8      i = lattice.l-1
9      for zeta in range(0,lattice.n):
10         derivatives[zeta] = {}
11         for phi in range(0,lattice.m)
       :
12             V_ffd =
       ffd_displacement_vector(geometry,
        lattice, physical_facet_tag, i,
       phi, zeta, deg=1)
13             shape_derivative_form =
       form(inner(V_ffd, normal) * G_neu
        * ds(physical_facet_tag))
14             eig = assemble_scalar(
       shape_derivative_form)
15             derivatives[zeta][phi] =
       eig
16     return derivatives
```

Listing 1 helmholtz-x code for computing shape derivatives of FFD control points. Line 6 represents the UFL form of Eq. (41). In the radial direction, we only compute the shape derivatives of the control points on the lateral surface. Between lines 9 and 15, we loop over the control points in the azimuthal (ϕ) and axial (z) directions, respectively.

We calculate the shape gradient aligned with the outward normal vector of the relevant control point. The physical interpretation of the complex-valued shape derivatives are shown in Fig. 11 with example design changes to reduce the growth rate of the eigenvalue.

In summary, the main steps of the adjoint based shape optimization method with FFD control points are:

- the three dimensional numerical grid is generated;
- the FFD lattice and its control points are defined after calculating the parametric coordinates of the nodes in the grid;
- direct and adjoint eigenmodes are calculated with P2 (degree 2) finite elements;
- the shape derivatives of the FFD control points are calculated and normalized;
- the shape is deformed in line with the direction provided by the normalized shape derivatives, with a certain step size.

4.3 Optimization

We calculate the direct and adjoint eigenmodes of the Rijke tube with helmholtz-x and obtain the shape derivatives of the control points on the lateral (Neumann) surface using Eq. (41). We only allow radial displacements of the control points and do not impose shape changes in the axial direction.

We then iterate over the control points on the lateral boundary and move them individually in the direction provided by the shape gradients. The deformed geometry of the Rijke tube is shown in Fig. 12. The growth rate of the eigenmode for the deformed design becomes negative after deformation. The general trend of the growth rate due to FFD changes is found to be similar to that in [23], in which B-Spline parametrization was applied. The example in this paper, however, allows radii changes for the inlet and outlet circular boundaries.

5 Conclusion

We present an open-source parallelized finite element framework, helmholtz-x, which solves the thermoacoustic Helmholtz equation, and present increasingly elaborate examples. In Sect. 2, we explain the FEM discretization and implementation details for helmholtz-x. In Sect. 3.1, we investigate axial eigenmodes in longitudinal combustors. We begin with a relatively simple example, the Rijke tube with Neumann boundary conditions. Then we propose a more detailed longitudinal example with area changes in the axial direction and choked boundary conditions at the inlet and outlet boundaries. We find that eigenmode computations of helmholtz-x in different configurations agree well with those of a network model for passive and active flame cases (Tables 2, 4). In Sect. 3.2, we present a numerical example of a laboratory 3D annular combustor, MICCA. We implement a 3D parabolic temperature field. We also present a state space representation

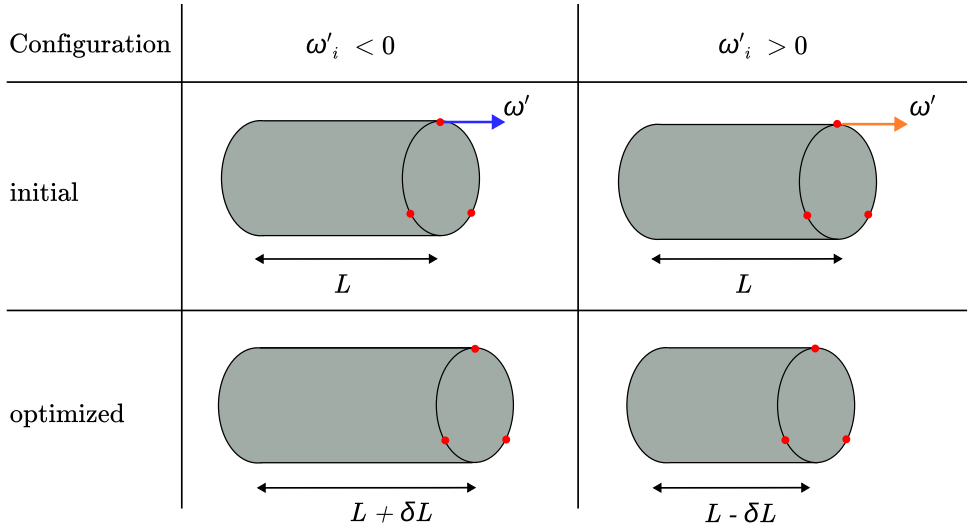


Fig. 11 Design changes that improve stability of the thermoacoustic system. ω' is the complex-valued shape derivative of the corresponding FFD control point. In scenarios in which the mode is unstable and the imaginary portion of the shape derivative at the control point has a negative sign, moving the control point along the outward normal vector direction improves system stability

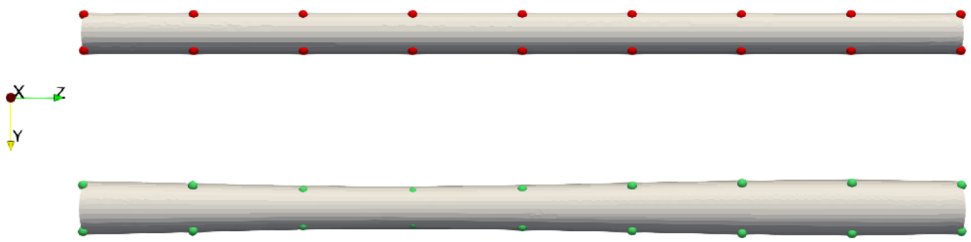


Fig. 12 Optimized geometry of the Rijke tube. The eigenfrequency after free form deformation is $\omega/2\pi = f = 202.171 - 0.354\ \mathrm{s}^{-1}$. The red (top) and green (bottom) dots are the initial and final positions of the FFD control points for the initial (top) and final (bottom) geometries after a few deformations

of the experimental data of the flame transfer function and obtain its analytical expression. Then we present different possible eigenmodes of the MICCA combustor. We visualize the corresponding eigenvectors in Fig. 9, in which helmholtz-x manages to capture axial (Fig. 9a), circumferential (e.g. Fig. 9c, d) and mixed modes (Fig. 9h). For the efficient calculation of circumferential modes, we also introduce Bloch boundary conditions to MICCA in Sect. 3.2.2. The circumferential eigenmode computations are much quicker with helmholtz-x than with existing 3D FEM tools in the literature. We also present parallelization capabilities of helmholtz-x for the eigenmode computations for the MICCA without Bloch BCs. Finally, we propose an adjoint based shape optimization application in Sect. 4. We present a free form deformation technique to parametrize the 3D Rijke tube geometry. We specify control points around the tube and calculate their shape derivatives using direct and adjoint eigenfunctions. In order to compute the shape gradients for each control point, the adjoint feature of helmholtz-x massively reduces the number of calculations compared with finite differences. Through this, helmholtz-x accelerates optimization procedures that can stabilize thermoacoustic systems, such as the Rijke tube (Fig. 12).

Given its applicability to the examples shown here, helmholtz-x could be a useful numerical tool to study and passively control thermoacoustic instabilities of complex shaped real-world combustors. The adjoint and parallel capabilities of helmholtz-x quickly calculate design changes that stabilize thermoacoustic systems. These can be combined with other constraints and entered into an optimization algorithm.

Several next steps are possible. Further acoustic or thermoacoustic test cases could be implemented, along with experimental or analytical analysis. More realistic geometries could be studied. The parallelization subroutines for the Bloch boundary conditions could be implemented. The acoustic damping caused by area changes could be implemented, in order to model acoustic dissipation in high speed incompressible flows. The acoustic properties of perforated liners could be accounted for, e.g. via Rayleigh conductivity. More robust interpolation schemes for 3D temperature fields could be implemented to incorporate the temperature distributions from experiments or LES. The FFD optimization

procedure could be advanced by including more complicated geometries and other engineering constraints.

Appendix A Further development of helmholtz-x

Appendix A.1 Customizable design

helmholtz-x can be developed further to include more acoustics and flame models. For example, Listing 2 shows how Robin boundary conditions are implemented. An acoustic liner model can be integrated through an impedance relation on the wall. This requires the block in Listing 2 to be duplicated, the name of the dictionary key to be updated, and the relation for R to be changed. helmholtz-x will then automatically include the new boundary condition when assembling matrix $\mathbf{B}$.

```
1 if 'Robin' in boundary_conditions[
    boundary]:
2     R = boundary_conditions[boundary]['
    Robin']
3     Z = (1+R)/(1-R)
4     integrals_Impedance = 1j * c / Z *
    inner(phi_k, phi_j) * ds(boundary)
5     integrals_R.append(
    integrals_Impedance)
```

Listing 2 Robin boundary condition implementation. The weak form in line 4 is identical to Eq. (14b). We add the contributions of the Robin boundaries to the list *integrals_R*.

If the user wants to impose an averaged value on a boundary, an approach similar to that in Listing 3 could be used. This example is for the implementation of choked boundaries where the heat capacity ratio on the boundary is averaged during the calculation of R.

```
1 if 'ChokedInlet' in
    boundary_conditions[boundary]:
2     A_inlet = MPI.COMM_WORLD.
    allreduce(assemble_scalar(form(
    AreaConstant * ds(boundary))), op
    =MPI.SUM)
3     gamma_inlet_form = form(gamma/
    A_inlet* ds(boundary))
4     gamma_inlet = MPI.COMM_WORLD.
    allreduce(assemble_scalar(
    gamma_inlet_form), op=MPI.SUM)
5
6     Mach = boundary_conditions[
    boundary]['ChokedInlet']
7     R = (1-gamma_inlet*Mach/(1+(
    gamma_inlet-1)*Mach**2))/(1+
    gamma_inlet*Mach/(1+(gamma_inlet
    -1)*Mach**2))
8     Z = (1+R)/(1-R)
9     integral_C_i = 1j * c / Z * inner
    (phi_k, phi_j) * ds(boundary)
10    integrals_R.append(integral_C_i)
```

Listing 3 Choked inlet boundary condition implementation. Lines 2 to 4 calculate the average γ on the choked boundary. Line 7 calculates the reflection coeffient for the choked inlet boundary using the near-upstream Mach number of the flow. Line 9 implements the Robin boundary condition with the specific impendance Z calculated in line 8. Finally the implemented Robin BC is added to the Robin integrals list in line 10.

The fluctuating body force ($\hat{f}_1$) and fluctuating mass ($\hat{m}_1$) can be used to model the damping caused by acoustic dissipation through area changes. The user needs to discretize Eq. (1) and include damping models in the terms $\hat{f}_1$ and/or $\hat{m}_1$. Manipulation of the UFL forms in Listing 4 adds these terms into matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$.

```
1  # Matrix A
2  a_form = form(-c**2* inner(grad(phi_k
       ), grad(phi_j))*dx)
3  A = assemble_matrix(a_form, bcs=
       bcs_Dirichlet)
4  A.assemble()
5  _A = A
6
7  # Matrix B
8  if integrals_R:
9      b_form = form(sum(integrals_R))
10     B = assemble_matrix(b_form)
11     B.assemble()
12     B_adj = B.copy()
13     B_adj.transpose()
14     B_adj.conjugate()
15     _B = B
16     _B_adj = B_adj
17
18 # Matrix C
19 c_form = form(inner(phi_k , phi_j) *
       dx)
20 C = assemble_matrix(c_form,
       bcs_Dirichlet)
21 C.assemble()
22 _C = C
```

Listing 4 UFL forms for construction of the acoustic matrices. Lines 2 and 19 represent Eqs. (14a) and (14c). In lines 3 and 20, Dirichlet boundary conditions are imposed. Lines 8 to 16 construct matrix **B** in Eq. (14) containing Robin boundary conditions using the list integrals_R.

Appendix A.2 Handling parallel sparse matrix data

This section contains details about the algorithms that obtain the sparse left and right vectors when constructing **D** and/or its adjoint. For the distributed flame matrix, the UFL forms of the left and right vectors are defined in listing 5.

```
1 left_form = form((gamma - 1) * q_0 /
      u_b * phi_k * h *  dx)
2 right_form = form(inner(n_r,grad(
      phi_j)) / rho * w * dx)
```

Listing 5 UFL forms for left and right vectors shown in Eq. (20).

We provide the vector assembly codes for the distributed flame matrix in Listing 6 and for the pointwise flame matrix in Listing 7.

```
1  def _assemble_vectors(self,
       problem_type='direct'):
2      left_vector = indices_and_values(
       left_form)
3      right_vector = indices_and_values
       (right_form)
4      if problem_type == 'direct':
5          left_vector =
       distribute_vector_as_chunks(
       left_vector)
6          right_vector =
       broadcast_vector(right_vector)
7      elif problem_type == 'adjoint':
8          right_vector =
       distribute_vector_as_chunks(
       right_vector)
9          left_vector =
       broadcast_vector(left_vector)
10     return left_vector, right_vector
```

Listing 6 We first define the left and right vector forms as in Listing 5 The internal function *indices_and_values* extracts the nonzero values from the left (line 2) and right (line 3) sparse vectors. If the problem is direct, the right vector is replicated over the processors and the left vector is distributed evenly. For the adjoint problem, the procedure is reversed.

```
1  def _assemble_vectors(self, flame, point):
2      left_form = form((gamma - 1) * q_0 / u_b * inner(h, phi_j)*dx(flame))
3      left_vector = indices_and_values(left_form)
4      _, _, owning_points, cell = determine_point_ownership( mesh._cpp_object, point, 1e-10)
5      right_vector = []
6      if len(cell) > 0: # Only add contribution if cell is owned
7          cell_geometry = mesh.geometry.x[mesh.geometry.dofmap[cell[0]], :gdim]
8          point_ref = mesh.geometry.cmaps[0].pull_back([point], cell_geometry)
9          right_form = Expression(inner(grad(TestFunction(V)), n_r), point_ref, comm=MPI.COMM_SELF)
10         dphij_x_rs = right_form.eval(mesh, cell)[0]
11         right_values = dphij_x_rs / rho_u
12         global_dofs = dofmaps.index_map.local_to_global(dofmaps.cell_dofs(cell[0]))
13         for global_dof, right_value in zip(global_dofs, right_values):
14             right_vector.append([global_dof, right_value ])
15     right_vector = broadcast_vector(right_vector)
16     return left_vector, right_vector
```

Listing 7 Code to calculate the nonzero data for the pointwise flame matrix. The parameters *flame* and *point* represent the flame tag and its measurement point (line 1). Line 2 is identical to the left vector of Eq. (21). For the left vector data, we use only the corresponding flame subdomain ($\mathbf{dx}_f$) during integration and we calculate its nonzero data in line 3. Line 9 is identical to the right vector of Eq. (21). We calculate the value of the gradient of the test function at the measurement point (line 11) and find the DOFs of the cell that includes the measurement point (line 12). We store the global indices of the DOFs of the cell and construct the (col index, value) pairs of the right vector (lines 13 and 14). Finally, we copy the data of the right vector over the processors for parallel pointwise **D** generation (line 15).

We present the MPI utility functions of helmholtz-x for constructing the sparse matrix **D** and its adjoint, $\mathbf{D}^H$. FEniCSx uses MPI for handling the parallelization [32]. When we parallelize the calculation using n_{proc} processes, FEniCSx partitions the mesh into n_{proc} pieces. Each piece has different nonzero entries for the left and right vectors that we use to construct **D** (Sect. 2.5). We need the positions and entries of those nonzero contributions for both vectors. We obtain these as (indices, nonzero values) pairs. For **D**, we obtain row indices from the left vector and column indices from the right vector. We calculate the nonzero entries by multiplying the nonzero values of the left and right vectors within the same process. The nonzero data for the right vector may, however, be 'None', while the contribution for the left vector may be nonzero. Without modification, the multiplication of nonzero entries in that process would give 'None', meaning that the unmodified algorithm would fail to insert some nonzero contributions during assembly. To address this, we implement a broadcasting function that copies the right vector data to each process, as shown in Listing 8.

```
1  def broadcast_vector(vector):
2      vector = MPI.COMM_WORLD.gather(vector, root=0)
3      if vector:
4          vector = [j for i in vector for j in i]
5      else:
6          vector=[]
7      vector = MPI.COMM_WORLD.bcast(vector,root=0)
8      return vector
```

Listing 8 Broadcasting function to gather the right vector indices and values from the processors to process 0 (line 2) and broadcast the nonzero contributions back to the processors (line 7) during a parallel run.

To improve the share of computational load of each process, we also distribute the left vector data evenly over the processors. The algorithm for this is in Listing 9.

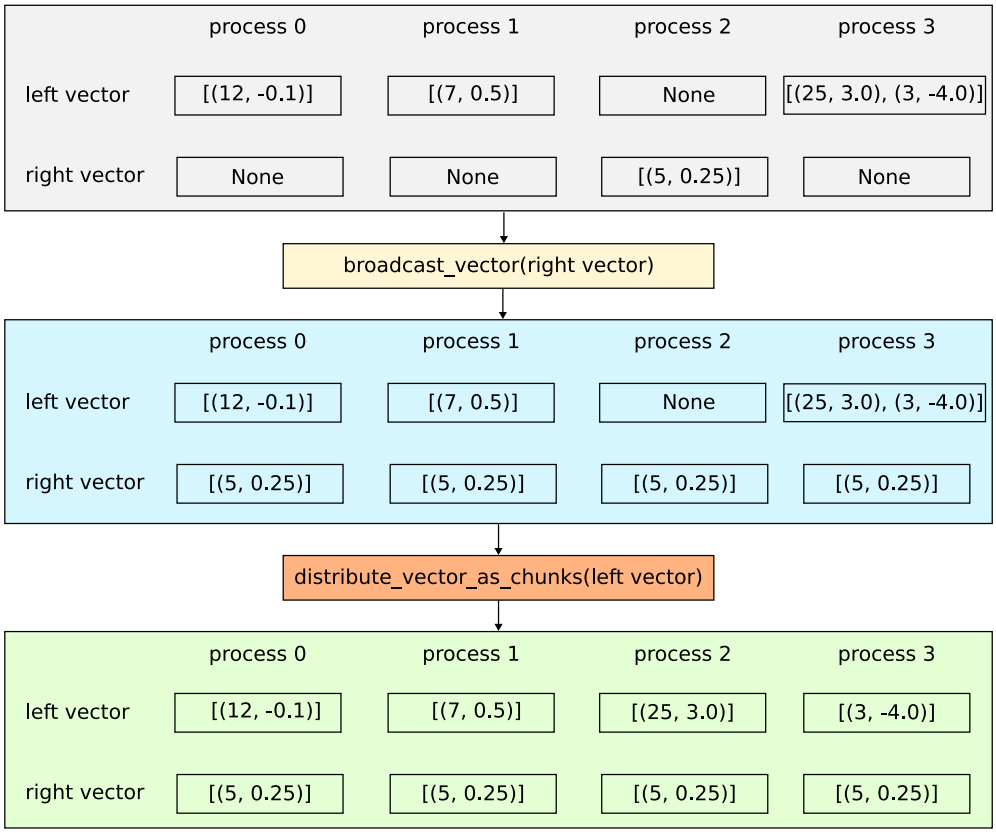


Fig. 13 Example of the nonzero data handling for left and right vectors using four processors. The left vector's data consists of nonzero row pairs (row indices, row values), while the right vector's data consists of nonzero column pairs (column indices, column values). When the left and right vectors contain non-zero data, the cross multiplication of the values of the vectors can give 'None', so nonzero contributions are lost (gray case). To prevent this, we copy the data of the right vector to each process (blue case) using the algorithm in Listing 8. We then evenly distribute the data of the left vector using the algorithm described in Listing 9. This shares the workload among processors (green case). Finally, the left and right vector data become ready for matrix construction

```
1  def distribute_vector_as_chunks(
       vector):
2      vector = MPI.COMM_WORLD.gather(
       vector, root=0)
3      if vector:
4          vector = [j for i in vector
       for j in i]
5          chunks = [[] for _ in range(
       MPI.COMM_WORLD.Get_size())]
6          for i, chunk in enumerate(
       vector):
7              chunks[i % MPI.COMM_WORLD
       .Get_size()].append(chunk)
8      else:
9          vector = None
10         chunks = None
11     vector = MPI.COMM_WORLD.scatter(
       chunks, root=0)
12     return vector
```

Listing 9 Function used to distribute the left vector indices and values over the processors during parallel runs. The algorithm initially gathers all nonzero data at the root (line 2). It then distributes the data across the processors as evenly as possible.

These operations are demonstrated in Fig. 13.

Appendix B Further test cases

Appendix B.1 Verification of passive acoustic eigenmodes

We demonstrate simple acoustic cases without a flame in order to compare the results of helmholtz-x against available analytical solutions. For these test cases, we repeat the calculations presented in Sect. 5 in [43]. We consider a two-dimensional acoustic domain with a length of $L = 0.4m$ and a height of $h = 0.1$m. We assume the speed of sound to be uniform, $c_0 = 450$m/s over the domain. We model the boundaries as a Neumann condition except for the top boundary that is modelled as an impedance (Robin) boundary condition. We define the impedance $Z = a + bi$, where a denotes the acoustic resistance and b denotes the acoustic reactance of the boundary. The manufactured solution for this case is [43]:

$$e^{2ik_y h}\left(k_y - \frac{k}{Z}\right) - \left(k_y + \frac{k}{Z}\right) = 0, \quad k_y = \sqrt{k^2 - \left(\frac{n\pi}{L}\right)^2}, \tag{B1}$$

where $k = \omega/c_0$ represents the wavenumber and n represents the mode number. We calculate the eigensolutions of Eq. (B1) with $n = 1$ for various purely reactance and purely resistance impedances. We set up these cases using helmholtz-x and compare the results in Fig. 14. For the presented numerical cases, the eigenfrequencies obtained with helmholtz-x show very good agreement with the analytical eigenfrequencies.

Appendix B.2 Verification of adjoint eigenmodes

We present tests to check the adjoint capability of helmholtz-x by replicating the results in [19]. The helmholtz-x codes for these cases are in the `numerical_examples/Longitudinal/PRF` folder in the repository. The parameters are in Table 7.

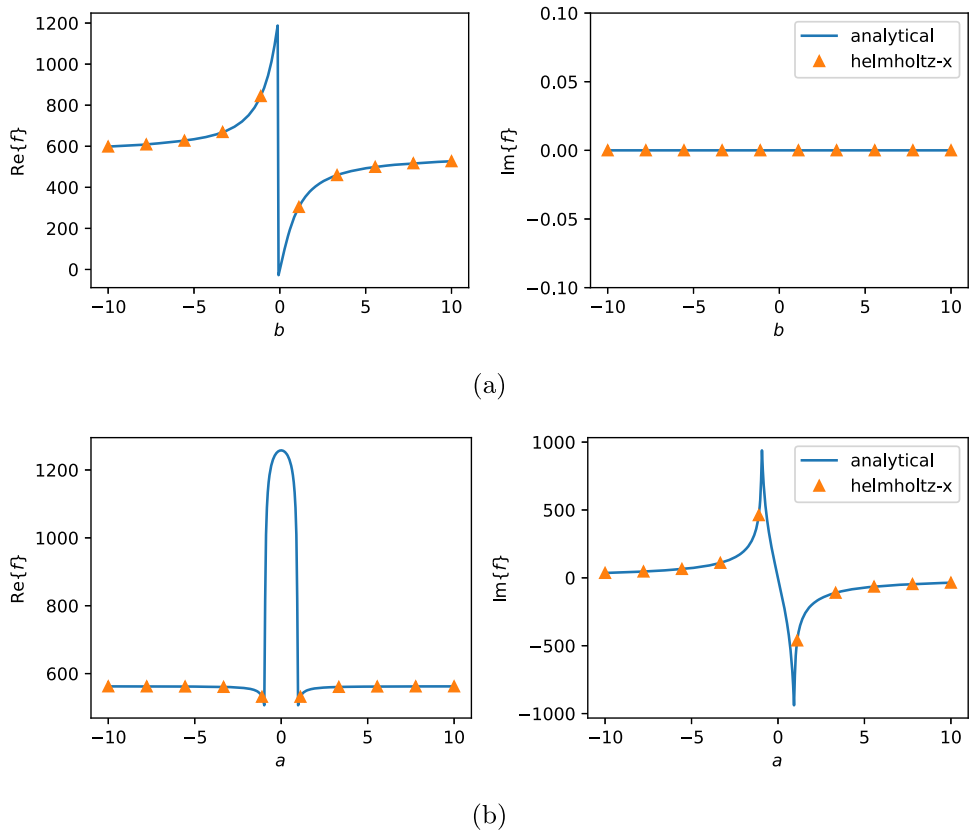


Fig. 14 Computed eigenfrequencies for the two-dimensional acoustic case with **a** purely reactive and **b** purely resistive impedance boundary conditions on $y = h$

We run test cases for multidimensional configurations (as in Sect. 3.1.1). The inlet and outlet boundaries are Robin, with reflections coefficients tabulated in Table. 7. The resulting eigenvalues are in Table 8.

We also show the direct and adjoint pressure eigenfunctions in Fig. 15.

Table 7 Dimensional parameters of the hot wire Rijke tube taken from [19]

Parameter	Value	Unit
L	1	m
d	0.047	m
r_{gas}	287.1	J kg^{-1} K^{-1}
p_0	100,000	Pa
ρ_u	1.22	kg m^{-3}
ρ_d	0.85	kg m^{-3}
q_0	200	W
u_b	0.1	m s^{-1}
n	1.4e-7	–
τ	0.0015	s
R_{inlet}	$-0.975 - 0.05$i	–
R_{outlet}	$-0.975 - 0.05$i	–
x_f	0.25	m
a_f	0.025	–
x_r	0.2	m
a_r	0.025	–

The interaction index n changes for 1D and 2D for dimensional consistency

Table 8 Eigenfrequencies of the passive and active flame test cases for the Rijke tube, where GR denotes the growth rate

Run	Direct f (1/s)	GR (rad/s)	Adjoint f (1/s)	GR (rad/s)
[19]	3.425513	+0.001926	3.425514	-0.001904
1D-helmholtz-x	3.421902	+0.002225	3.421902	-0.002224
2D-helmholtz-x	3.422663	+0.002180	3.422663	-0.002180
3D-helmholtz-x	3.420690	+0.002666	3.420690	-0.002667

The grid resolutions of the helmholtz-x tests can be improved to obtain eigenfrequencies closer to the results in [19]

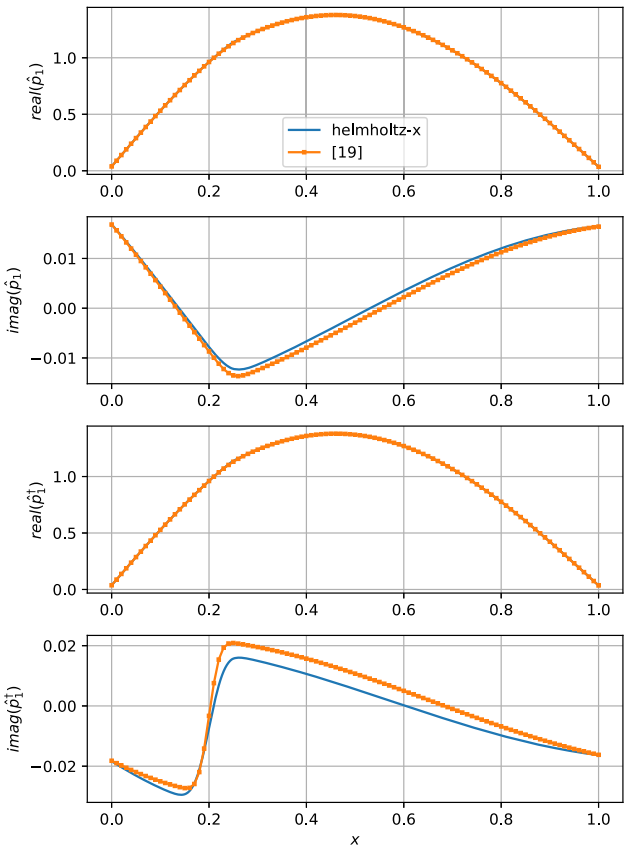


Fig. 15 1D direct and adjoint pressure eigenfunctions in the Rijke tube using helmholtz-x and [19]. The interval is discretized into 100 uniform sections and a first order continuous Lagrange space has been used for the simulation using the reference [19]

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00366-025-02107-1.

Acknowledgements Ekrem Ekici thankfully acknowledges doctoral funding from Türkiye's Ministry of National Education. Stefano Falco acknowledges the European Union's Framework Programme for Research and Innovation Horizon 2020 under the Marie Skłodowska Curie grant agreement ANNULIGhT No 765998. The authors would like to thank Simon Stow for supplying the network code and advising on its use. The authors are also grateful to Davide Laera for providing us the experimental flame transfer function data of the MICCA combustor.

Author Contributions Ekrem Ekici: Conceptualization, Methodology, Software, Visualization, Data curation, Writing. Stefano Falco: *Writing.* Matthew P. Juniper: Supervision, Methodology, Writing.

Data availability The source code of helmholtz-x is available on the GitHub repository https://github.com/ekremekc/helmholtz-x/tree/paper.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Crocco L, Cheng S-I (1956) Theory of combustion instability in liquid propellant rocket motors. Advisory Group for Aeronautical Research and Development, North Atlantic Treaty Organization. https://doi.org/10.1017/S0022112057210774
2. Dowling AP, Stow SR (2003) Acoustic analysis of gas turbine combustors. J Propul Power 19(5):751–764. https://doi.org/10.2514/2.6192
3. Li J, Yang D, Luzzato C, Morgans AS (2015) Open source combustion instability low order simulator (OSCILOS-Long) technical report. Technical report, Imperial College London, London. https://oscilos.com/
4. Leandro R, Huber A, Polifke W (2010) TaX-a low-order modeling tool for thermo-and aero-acoustic instabilities. Technical report, Professur für Thermofluiddynamik. https://mediatum.ub.tum.de/1249319
5. Mensah GA, Campa G, Moeck JP (2016) Efficient computation of thermoacoustic modes in industrial annular combustion chambers based on Bloch-wave theory. J Eng Gas Turbines Power 138(8):081502. https://doi.org/10.1115/1.4032335
6. Mensah GA (2019) Efficient computation of thermoacoustic modes. PhD thesis, Technische Universität Berlin. https://doi.org/10.14279/depositonce-8952
7. Mensah GA, Magri L, Moeck JP (2018) Methods for the calculation of thermoacoustic stability boundaries and Monte Carlo-free uncertainty quantification. J Eng Gas Turbines Power 140(6):061501. https://doi.org/10.1115/1.4038156
8. Buschmann PE, Mensah GA, Nicoud F, Moeck JP (2019) Solution of thermoacoustic eigenvalue problems with a non-iterative method. Turbo expo: power for land, sea, and air, vol 58615. American Society of Mechanical Engineers, USA, pp 04–04051
9. Mensah GA, Orchini A, Buschmann PE, Grubišić L (2022) A subspace-accelerated method for solving nonlinear thermoacoustic eigenvalue problems. J Sound Vib 520:116553. https://doi.org/10.1016/j.jsv.2021.116553
10. Mensah GA, Buschmann PE, Orchini A (2022) Iterative solvers for the thermoacoustic nonlinear eigenvalue problem and their convergence properties. Int J Spray Combust Dyn 14(1–2):30–41. https://doi.org/10.1177/17568277221084464
11. Silva CF, Prieto L, Ancharek M, Marigliani P, Mensah GA (2021) Adjoint-based calculation of parametric thermoacoustic maps of an industrial combustion chamber. J Eng Gas Turbines Power 143(1):011003. https://doi.org/10.1115/1.4049295
12. Motheau E, Nicoud F, Poinsot T (2014) Mixed acoustic-entropy combustion instabilities in gas turbines. J Fluid Mech 749:542–576. https://doi.org/10.1017/jfm.2014.245
13. Martin CE, Benoit L, Sommerer Y, Nicoud F, Poinsot T (2006) Large-eddy simulation and acoustic analysis of a swirled staged

turbulent combustor. AIAA J 44(4):741–750. https://doi.org/10.2514/1.14689
14. Boudier G, Lamarque N, Staffelbach G, Gicquel L, Poinsot T (2009) Thermo-acoustic stability of a helicopter gas turbine combustor using large eddy simulation. Int J Aeroacoust 8(1):69–93. https://doi.org/10.1260/147547209786234975
15. Poinsot T (2017) Prediction and control of combustion instabilities in real engines. Proc Combust Inst 36(1):1–28. https://doi.org/10.1016/j.proci.2016.05.007
16. Juniper MP, Sujith RI (2018) Sensitivity and nonlinearity of thermoacoustic oscillations. Annu Rev Fluid Mech 50:661–689. https://doi.org/10.1146/annurev-fluid-122316-045125
17. Magri L, Juniper MP (2013) Sensitivity analysis of a time-delayed thermo-acoustic system via an adjoint-based approach. J Fluid Mech 719:183–202. https://doi.org/10.1017/jfm.2012.639
18. Aguilar JG, Magri L, Juniper MP (2017) Adjoint-based sensitivity analysis of low-order thermoacoustic networks using a wave-based approach. J Comput Phys 341:163–181. https://doi.org/10.1016/j.jcp.2017.04.013
19. Juniper MP (2018) Sensitivity analysis of thermoacoustic instability with adjoint Helmholtz solvers. Phys Rev Fluids 3(11):110509. https://doi.org/10.1103/PhysRevFluids.3.110509
20. Magri L (2019) Adjoint methods as design tools in thermoacoustics. Appl Mech Rev 71(2):020801. https://doi.org/10.1115/1.4042821
21. Aguilar JG, Juniper MP (2020) Thermoacoustic stabilization of a longitudinal combustor using adjoint methods. Phys Rev Fluids 5(8):083902. https://doi.org/10.1103/PhysRevFluids.5.083902
22. Aguilar JG, Juniper MP (2018) Adjoint methods for elimination of thermoacoustic oscillations in a model annular combustor via small geometry modifications. Turbo expo: power for land, sea, and air, vol 51050. American Society of Mechanical Engineers, USA, pp 04–04054
23. Falco S, Juniper MP (2021) Shape optimization of thermoacoustic systems using a two-dimensional adjoint Helmholtz solver. J Eng Gas Turbines Power 143(7):071025. https://doi.org/10.1115/1.4049305
24. Ekici E, Falco S, Juniper MP (2024) Shape sensitivity of thermoacoustic oscillations in an annular combustor with a 3D adjoint Helmholtz solver. Comput Methods Appl Mech Eng 418:116572. https://doi.org/10.1016/j.cma.2023.116572
25. Sederberg TW, Parry SR (1986) Free-form deformation of solid geometric models. In: Proceedings of the 13th annual conference on computer graphics and interactive techniques, pp 151–160. https://doi.org/10.1145/15922.15903
26. Samareh JA (2004) Aerodynamic shape optimization based on free-form deformation. In: 24th international congress of the aeronautical sciences. https://doi.org/10.2514/6.2004-4630
27. He X, Li J, Mader CA, Yildirim A, Martins JR (2019) Robust aerodynamic shape optimization—from a circle to an airfoil. Aerosp Sci Technol 87:48–61. https://doi.org/10.1016/j.ast.2019.01.051
28. Giugno A, Shahpar S, Traverso A (2020) Adjoint-based optimization of a modern jet-engine fan blade. Turbo expo: power for land, sea, and air, vol 84096. American Society of Mechanical Engineers, USA, pp 02–38026
29. John A, Shahpar S, Qin N (2017) Novel compressor blade shaping through a free-form method. J Turbomach 139(8):081002. https://doi.org/10.1115/1.4035833
30. Li L, Yuan T, Li Y, Yang W, Kang J (2019) Multidisciplinary design optimization based on parameterized free-form deformation for single turbine. AIAA J 57(5):2075–2087. https://doi.org/10.2514/1.J057819
31. Barrata IA, Dean JP, Dokken JS, Habera M, Hale J, Richardson C, Rognes ME, Scroggs MW, Sime N, Wells GN (2023) DOLFINx: the next generation FEniCS problem solving environment. https://doi.org/10.5281/zenodo.10447666
32. Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J (1998) MPI-the complete reference, volume 1: the MPI core, 2nd (revised) edn. MIT Press, Cambridge, MA, USA. https://dl.acm.org/doi/10.5555/552013
33. Dalcín L, Paz R, Storti M (2005) MPI for Python. J Parallel Distrib Comput 65(9):1108–1115. https://doi.org/10.1016/j.jpdc.2005.03.010
34. Alnæs MS (2012) UFL: a finite element form language. Automated solution of differential equations by the finite element method: the FEniCS Book. Springer, Berlin, Heidelberg, pp 303–338
35. Logg A, Ølgaard KB, Rognes ME, Wells GN (2012) FFC: the FEniCS form compiler. In: Automated solution of differential equations by the finite element method: the FEniCS Book, pp 227–238. https://doi.org/10.1007/978-3-642-23099-8_11
36. Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, Dalcin L, Dener A, Eijkhout V, Gropp W, Karpeyev D, Kaushik D, Knepley M, May D, McInnes LC, Mills R, Munson T, Rupp K, Sanan P, Smith B, Zampini S, Zhang H, Zhang H (2020) PETSc users manual (rev. 3.13). https://doi.org/10.2172/1614847
37. Dalcin LD, Paz RR, Kler PA, Cosimo A (2011) Parallel distributed computing using python. Adv Water Resour 34(9):1124–1139. https://doi.org/10.1016/j.advwatres.2011.04.013
38. Hernandez V, Roman JE, Vidal V (2005) SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans Math Software (TOMS) 31(3):351–362. https://doi.org/10.1145/1089014.1089019
39. Falco S (2022) Shape optimization for thermoacoustic instability with an adjoint Helmholtz solver. PhD thesis, University of Cambridge. https://doi.org/10.17863/CAM.84402
40. Crocco L, Grey J, Harrje DT (1960) Theory of liquid propellant rocket combustion instability and its experimental verification. ARS J 30(2):159–168. https://doi.org/10.2514/8.5020
41. Rienstra SW, Hirschberg A (2004) An introduction to acoustics. https://www.win.tue.nl/~sjoerdr/papers/boek.pdf
42. Bloch F (1929) Über die quantenmechanik der elektronen in kristallgittern. Z Phys 52(7–8):555–600. https://doi.org/10.1007/BF01339455
43. Nicoud F, Benoit L, Sensiau C, Poinsot T (2007) Acoustic modes in combustors with complex impedances and multidimensional active flames. AIAA J 45(2):426–441. https://doi.org/10.2514/1.24933
44. Ahmed Z, Kinjol FJ, Ananya IJ (2021) Comparative analysis of six programming languages based on readability, writability, and reliability. In: 2021 24th International conference on computer and information technology (ICCIT). IEEE, pp 1–6. https://doi.org/10.1109/ICCIT54785.2021.9689813
45. Geuzaine C, Remacle J-F (2009) Gmsh: a 3D finite element mesh generator with built-in pre-and post-processing facilities. Int J Numer Meth Eng 79(11):1309–1331. https://doi.org/10.1002/nme.2579
46. Ayachit U (2015) The ParaView Guide: a parallel visualization application. Kitware Inc., Clifton Park, NY, USA
47. Bourgouin J-F, Durox D, Moeck JP, Schuller T, Candel S (2015) Characterization and modeling of a spinning thermoacoustic instability in an annular combustor equipped with multiple matrix injectors. J Eng Gas Turbines Power 137(2):021503. https://doi.org/10.1115/1.4028257
48. Laera D, Schuller T, Prieur K, Durox D, Camporeale SM, Candel S (2017) Flame describing function analysis of spinning and standing modes in an annular combustor and comparison with experiments. Combust Flame 184:136–152. https://doi.org/10.1016/j.combustflame.2017.05.021
49. Durox D, Bourgouin J-F, Moeck JP, Philip M, Schuller T, Candel S (2013) Nonlinear interactions in combustion instabilities coupled by azimuthal acoustic modes. In: n3l-Int'l Summer school and workshop on non-normal and nonlinear effects in aero-and

thermoacoustics, p 14. https://mediatum.ub.tum.de/doc/1253524/document.pdf
50. Bourgouin J-F, Durox D, Moeck J, Schuller T, Candel S (2015) A new pattern of instability observed in an annular combustor: the slanted mode. Proc Combust Inst 35(3):3237–3244. https://doi.org/10.1016/j.proci.2014.06.029
51. Mensah GA, Magri L, Orchini A, Moeck JP (2019) Effects of asymmetry on thermoacoustic modes in annular combustors: a higher-order perturbation study. J Eng Gas Turbines Power 141(4):041030. https://doi.org/10.1115/1.4041007
52. Gustavsen B, Semlyen A (1999) Rational approximation of frequency domain responses by vector fitting. IEEE Trans Power Delivery 14(3):1052–1061. https://doi.org/10.1109/61.772353
53. Yadav R, Lee W, Elibol M, Papadakis M, Lee-Patti T, Garland M, Aiken A, Kjolstad F, Bauer M (2023) Legate sparse: distributed sparse computing in python. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, pp 1–13. https://doi.org/10.1145/3581784.3607033
54. Hiremath V, Roman JE (2023) Acoustic modal analysis with heat release fluctuations using nonlinear eigensolvers. Appl Math Comput 458:128249. https://doi.org/10.1016/j.amc.2023.128249