Tutorial 1: Obtaining thermoacoustic eigenvalue sensitivities with adjoint methods

Matthew P. Juniper
Engineering Department, University of Cambridge, UK

Contents

1. Overview

This tutorial starts from the acoustic momentum and energy equations for a 1D thermoacoustic system with zero mean flow. Four solution methods are described and coded as Matlab functions: a travelling wave method (`fun_travwave.m`), a Finite Difference Helmholtz method (`fun_Helm_FD.m`), a Finite Element Helmholtz method (`fun_Helm_FE.m`), and a Galerkin method (`fun_Galerkin.m`). Small differences between their solutions arise due to the different approximations in each method. The functions are overloaded such that the sensitivities of the eigenvalue with respect to the model parameters can also be returned. These sensitivities are calculated with adjoint methods. All figures and tables in the tutorial are generated with the Matlab functions supplied with the tutorial.

2. The model and its governing equations

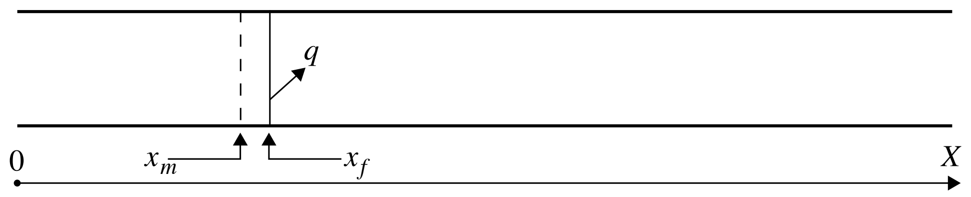

Figure 1

Diagram of the open-ended tube extending from 0 (upstream) to X (downstream). The heat release, q, occurs at position x_f and is a function of the acoustic velocity at x_m. The mean flow, which is defined as positive in the positive x-direction, is set to zero.

We consider one-dimensional oscillations in an open-ended tube extending from 0 to

X containing gas at uniform density, $\overline{\rho}$, uniform pressure, $\overline{p}$, and uniform ratio of specific heats, γ. We set the mean flow to zero and consider planar acoustic perturbations to the velocity, u, and pressure, p. A heat source is placed at $x = x_f$, with heat release rate $q(t) = nu(x_m, t-\tau)$ Watts per m^2, where n is a real constant with units of Joules per m^3, τ is a time delay, and x_m is the position at which u is measured. We neglect the mean density drop across the heat source, and viscous and thermal dissipation. The dimensional acoustic momentum and energy equations are:

$$\overline{\rho}\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = 0 \tag{1a}$$
$$\frac{\partial p}{\partial t} + \gamma\overline{p}\frac{\partial u}{\partial x} = (\gamma-1)q\delta_D(x-x_f) \tag{1b}$$

where δ_D is the Dirac delta. The speed of sound is $\overline{c} \equiv \sqrt{\gamma\overline{p}/\overline{\rho}}$. We define a reference length, $L_{ref} = X$, a reference speed, $U_{ref} = \overline{c}$, and a reference pressure, $P_{ref} = \overline{p}$. The non-dimensional momentum and energy equations are:

$$\gamma\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = 0 \tag{2a}$$
$$\frac{\partial p}{\partial t} + \gamma\frac{\partial u}{\partial x} = (\gamma-1)q\delta_D(x-x_f) \tag{2b}$$

and it is convenient to define $\gamma' \equiv (\gamma-1)/\gamma$. The model parameters are n, τ, x_m, and x_f.

3. Four methods to solve the governing equations

This tutorial demonstrates four numerical methods commonly used to solve thermoacoustic governing equations. They are applied to (2) in order to demonstrate the principles behind each method. When more elaborate models are considered, the codes become more intricate but the principals remain the same.

3.1. The travelling wave method (`fun_travwave.m`)

Integrating (2) in x and assuming no accumulation within a control volume enclosing the heat source at x_f leads to two jump conditions:

$$[p]_{x_f^-}^{x_f^+} = 0 \tag{3a}$$
$$[u]_{x_f^-}^{x_f^+} = \gamma' nu(x_m, t-\tau) \tag{3b}$$

We consider the acoustic standing wave to be the sum of a forwards-travelling wave $f(x,t) = f(x-t)$, and a backwards-travelling wave $g(x,t) = g(t+x)$, such that $p = f + g$ and $u = (f-g)/\gamma$. These automatically satisfy (2) upstream and downstream of the heater, where $q = 0$. We assume a single frequency wave and perform a Laplace transform on $f(x-t)$ to obtain $f(x,t) = F(x,s)\mathrm{e}^{st} = F_0\mathrm{e}^{-s(x-x_0)}\mathrm{e}^{st}$, where $F_0 \equiv f(x_0, 0)$ and s is complex. Similarly, $g(x,t) = G(x,s)\mathrm{e}^{st} = G_0\mathrm{e}^{+s(x-x_0)}\mathrm{e}^{st}$. Backwards-travelling waves reflect off the upstream boundary, with reflection coefficient R_u, and return as forward-travelling waves time $\tau_u \equiv 2x_f$ later. This enforces $F_{f-} = R_u G_{f-}\mathrm{e}^{-s\tau_u}$ just upstream of the heat source at x_f. A similar expression is formed just downstream of the heat source with the downstream reflection coefficient, R_d. In this tutorial, R_u and R_d are set to -1. The measurement point, x_m, lies upstream of the heat source, x_f, so we define another time delay $\tau_{mf} \equiv (x_f - x_m)$. For convenience, we define five internal parameters **(h,t,tu,td,tmf)** from the four model parameters (n, τ, x_m, x_f):

```
% h = (gamma-1)/gamma * n
h = (param.gam-1)/param.gam*param.n;
% time delay
t = param.tau;
% tu, time for wave to travel flame -> upstream boundary -> flame
tu = 2*(param.x_f);
% td, time for wave to travel flame -> downstream boundary -> flame
td = 2*(1-param.x_f);
% tmf, time for wave to travel from flame -> measurement point
tmf = (param.x_f - param.x_m);
```

Listing 1 **Definitions of the internal parameters for** `fun_travwave.m`

We eliminate F_{f-} and G_{f+} with the boundary conditions, define $\phi \equiv \left(\mathrm{e}^{-s\tau_{mf}} - R_u \mathrm{e}^{+s\tau_{mf}} \mathrm{e}^{-s\tau_u}\right)$, and express the jump conditions (3) in matrix form:

$$\mathbf{L}(s)\mathbf{q} \equiv \begin{bmatrix} (1 + R_u \mathrm{e}^{-s\tau_u}) & -(1 + R_d \mathrm{e}^{-s\tau_d}) \\ (1 - R_u \mathrm{e}^{-s\tau_u}) + (\gamma' n \mathrm{e}^{-s\tau})\phi & (1 - R_d \mathrm{e}^{-s\tau_d}) \end{bmatrix} \begin{bmatrix} G_{f-} \\ F_{f+} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4}$$

This is coded as an embedded function `fun_L.m` (which also permits R to vary):

```
function [L] = fun_L(s,h,t,tu,td,tmf,R)
L = [ ...
 1+R*exp(-s*tu), ...
-1-R*exp(-s*td); ...
 1-R*exp(-s*tu)+h*exp(-s*t)*(exp(-s*tmf)-R*exp(+s*tmf)*exp(-s*tu)), ...
 1-R*exp(-s*td) ...
    ];
end
```

Listing 2 `fun_L` **: embedded function that creates matrix $\mathbf{L}(s)$**

For the iteration algorithm, we also require $\partial\mathbf{L}/\partial s$, which is derived by hand and coded as another embedded function:

```
function [dLds] = fun_dLds(s,h,t,tu,td,tmf,R)
dLds = [...
-R*tu*exp(-s*tu), ...
 R*td*exp(-s*td); ...
 R*tu*exp(-s*tu) ...
-h*(exp(-s*tmf)-R*exp(+s*tmf)*exp(-s*tu))*t*exp(-s*t) ...
+h*(-tmf*exp(-s*tmf)-R*tmf*exp(+s*tmf)*exp(-s*tu) ...
+R*tu*exp(+s*tmf)*exp(-s*tu))*exp(-s*t), ...
 R*td*exp(-s*td) ...
       ];
end
```

Listing 3 `fun_dLds` **: embedded function that creates matrix $\partial\mathbf{L}/\partial s$**

We then find s for which the determinant of $\mathbf{L}(s)$ equals zero. These are the eigenvalues, s, with corresponding eigenvectors, $\mathbf{q} \equiv [G_{f-}, F_{f+}]^T$, and are found with a Newton method that uses Jacobi's formula:

```
% Set tolerance, initial s, and dummy ds
tol = 1e-8; s = scheme.s0; dels = 2*tol;
while abs(dels) > tol
```

```
    % Evaluate L
    L = fun_L(s,h,tu,td,tmf,t,R);
    % Evaluate dLds
    dLds = fun_dLds(s,h,tu,td,tmf,t,R);
    % evaluate new s with Jacobi's formula
    dels = - 1/trace(L\dLds);
    % Update s
    s = s + dels;
end
% Evaluate the right eigenvector, q_dir, such that L*q_dir = [0;0]
q_dir = [+1+R*exp(-s*td) ; +1+R*exp(-s*tu)];
```
Listing 4 Newton method to find roots of $\mathbf{L}(s)$ and the corresponding eigenvector

The eigenfunctions in physical space are obtained by extracting the eigenvector $[G_{f-}, F_{f+}]^T$, evaluating F_{f-} and G_{f+} from the reflection coefficients, and substituting these into $P(x,s) = F(x,s) + G(x,s) = F_{f-}\mathrm{e}^{-s(x-x_f)} + G_{f-}\mathrm{e}^{+s(x-x_f)}$ and $U(x,s) = (F(x,s) - G(x,s))/\gamma = (F_{f-}\mathrm{e}^{-s(x-x_f)} - G_{f-}\mathrm{e}^{+s(x-x_f)})/\gamma$ upstream of the heat source, and into similar expressions downstream of the heat source.

3.2. The Helmholtz Finite Difference method (`fun_Helm_FD.m`)

Combining equations (2a) and (2b) and performing Laplace transforms, $p(x,t) = P(x)\mathrm{e}^{st}$, gives: $s^2 P - P'' = (\gamma' n \mathrm{e}^{-s\tau})\delta_D(x - x_f)P'(x_m)$, where $P' \equiv \mathrm{d}P/\mathrm{d}x$. In this method, the problem is better posed if the heat is released over a region of space, $v(x)$, centred on x_f, and if the reference velocity, $U(x_m)$, is measured over a region of space, $w(x)$, centred on x_m. The governing equation then becomes $s^2 P - P'' = (\gamma' n \mathrm{e}^{-s\tau})v(x)\int P' w(\xi)\mathrm{d}\xi$, where $\int v(x)\mathrm{d}x = 1$ and $\int w(\xi)\mathrm{d}\xi = 1$. In the finite difference framework, $P(x)$ is discretized onto Gauss–Lobatto spaced gridpoints and its values held in the column vector $\mathbf{P}$. A difference matrix $\mathbf{D}$ is formed such that $\mathbf{P}'$ is given by $\mathbf{DP}$. Similarly, a mass matrix $\mathbf{M}$ is formed such that $\int fg\,\mathrm{d}x = \mathbf{f}^T\mathbf{M}\mathbf{g}$. The discretized governing equation is:

$$\mathbf{L}(s)\mathbf{q} \equiv \left(s^2\mathbf{I} + \mathrm{e}^{-s\tau}\mathbf{F} - \mathbf{D}^2\right)\mathbf{P} = 0, \tag{5}$$

where $\mathbf{F} \equiv \gamma' n \mathbf{v}\mathbf{w}^T\mathbf{M}\mathbf{D}$ and $\mathbf{v}$ and $\mathbf{w}$ are column vectors containing the values of $v(x)$ and $w(x)$ at the gridpoints. For the tutorial, $v(x)$ and $w(x)$ are both Gaussian distributions with the same width. The internal parameters (`F`,`t`) are expressed in terms of the four model parameters (n, τ, x_m, x_f):

```
% Generate the heat release envelope, v(x), which integrates to 1
v = exp(-(x-param.x_f).^2/param.an^2)/sqrt(pi)/param.an;
% Generate the measurement envelope, w(x), which integrates to 1
w = exp(-(x-param.x_m).^2/param.an^2)/sqrt(pi)/param.an;
% Wrap v, w, gamma, param.n, M, and D into a matrix, F
F = (v*w') * (param.gam-1)/param.gam * param.n * M * D;
% Extract tau
t = param.tau;
```
Listing 5 Definitions of the internal parameters for `fun_Helm_FD.m`

The same Newton method is used, requiring $\mathbf{L}$ and $\partial\mathbf{L}/\partial s$:

```
function [L] = fun_L(D2,I,F,t,s)
    L    = s^2*I + exp(-s*t) * F - D2 ;
end

```

N	`fun_travwave`	`fun_Helm_FD`	`fun_Helm_FE`	`fun_Galerkin`
–	0.12188+3.22278i	–	–	–
1	–	–	–	0.13576+3.21473i
10	–	0.00000+3.14159i	0.00000+3.15453i	0.12301+3.22226i
40	–	0.06990+3.18462i	0.10457+3.21003i	0.12363+3.22196i
100	–	0.12303+3.22356i	0.12181+3.22286i	0.12115+3.22310i
400	–	0.12182+3.22274i	0.12182+3.22275i	0.12206+3.22270i

$x_m = 0.20000$, $x_f = 0.25000$, $n = 1.00000$, $\tau = 0.34157$, $\gamma = 1.40000$, $R = -1.00000$

Table 1 Eigenvalues, s, calculated with four different methods and, where relevant, up to five different resolutions, N, for the same thermoacoustic system. The growth rate is s_r and the frequency is s_i. The solutions approach each other as the resolution increases. (`Tab_comparisons.m`)

```
function [dLds] = fun_dLds(I,F,t,s)
    dLds = 2*s*I - t * exp(-s*t) * F ;
end
```

Listing 6 Calculation of L and $\partial\mathbf{L}/\partial s$ for `fun_Helm_FD.m`

Homogenous Dirichlet boundary conditions on P are applied at both ends by removing the outer rows and columns of the matrix. (For a closed tube, Homogenous Neumann conditions are required and, for a tube with $|R| \neq 1$, Robin conditions are required.) Once the eigenvalue, s, and the deficient matrix $\mathbf{L}(s)$ have been calculated, the eigenvectors are found from the null space of $\mathbf{L}$:

```
% Set tolerance, initial s, and dummy dels
tol = 1e-8; s = scheme.s0; dels = 2*tol;
while abs(dels) > tol
    % Evaluate L and apply Dirichlet boundary conditions
    L = fun_L(D2,I,F,t,s); L = L(2:N,2:N);
    % Evaluate dL/ds and apply Dirichlet boundary conditions
    dLds = fun_dLds(I,F,t,s); dLds = dLds(2:N,2:N);
    % evaluate new s with QR decomposition (H. Yu) and Jacobi's formula
    [Q,R] = qr(L); dels = - 1/trace(R\(Q'*dLds));
    % Update s
    s = s + dels;
end
% Find the corresponding eigenvectors for P
L = fun_L(D2,I,F,t,s); L = L(2:N,2:N);
P_dir = null(L);  P_dir = [0;P_dir;0];
```

Listing 7 Finding the eigenmode with homogenous Dirichlet boundary conditions using a Newton method with Jacobi's formula

3.3. The Helmholtz Finite Element method (`fun_Helm_FE.m`)

The governing equations and solution method for the Helmholtz Finite Element method are identical to those of the Helmholtz Finite Difference method (section 3.2) but the matrices differ.

3.4. The Galerkin method (`fun_Galerkin.m`)

The acoustic velocity and pressure are expressed as sums of the acoustic modes of the system for zero heat release with homogenous Dirichlet boundary conditions:

$$u(x,t) = \sum_{j=1}^{N} u_j(t)\cos(j\pi x) \tag{6a}$$

$$p(x,t) = \sum_{j=1}^{N} p_j(t)\sin(j\pi x) \tag{6b}$$

Substituting (6) into (2a) gives $p_j = -(\gamma/j\pi)\dot{u}_j$. Substituting these expressions into (2b), multiplying by $\sin(k\pi x)$, and integrating over the domain, gives:

$$-\ddot{u}_k - (k\pi)^2 u_k = \gamma' n v_k \sum_j u_j(t-\tau) w_j \tag{7}$$

where $v_k \equiv 2k\pi\sin(k\pi x_f)$ and $w_j \equiv \cos(j\pi x_m)$. This is a set of delay differential equations (DDEs) in the time domain for u_k. Performing the Laplace transform $u_k(t) = U_k \mathrm{e}^{st}$ leads to

$$\mathbf{L}(s)\mathbf{q} \equiv \left(s^2\mathbf{I} + \mathrm{e}^{-s\tau}\mathbf{F} - \mathbf{D}^2\right)\mathbf{U} = 0, \tag{8}$$

where $\mathbf{F} \equiv \gamma' n \mathbf{v}\mathbf{w}^T$, $\mathbf{v}$ and $\mathbf{w}$ are column vectors containing v_k and w_j, $\mathbf{D}^2$ is a second order difference matrix containing $-(k\pi)^2$ along the diagonal, and $\mathbf{U}$ is a column vector of U_k. The internal parameters (`F,t`) are expressed in terms of the four model parameters:

```matlab
% Create a column vector pi*(1:N)
kpi = pi*(1:N)';
% Create the heat release envelope v(x)
v = (2*kpi) .* sin(kpi*param.x_f);
% Create the measurement envelope w(x)
w = cos(kpi*param.x_m);
% Wrap v, w, gamma, param.n into a matrix, F
F = (v*w') * (param.gam-1)/param.gam * param.n;
% Extract tau
t = param.tau;
```

Listing 8 Definitions of the internal parameters for `fun_Galerkin.m`

The solution method is then identical to that of the Helmholtz Finite Difference method (section 3.2).

3.5. Results

Table 1 shows the eigenvalue, s, calculated at various resolutions with the four different methods. Figure 2 shows the corresponding $P(x)$ and $U(x)$ eigenvectors for the highest resolution case of each method. The eigenvectors differ only around the heat release zone, at $x = x_f = 0.25$. These differences arise because (i) the Galerkin method suffers from Gibbs fringes when it attempts to form the discontinuity in $U(x)$ and (ii) for the Helmholtz solvers, the heat release and measurement zones are spread out in space.

4. Eigenvalue sensitivity

The principles of adjoint-based sensitivity can be explained in terms of the nonlinear eigenvalue problem $\mathbf{L}(s)\mathbf{q} = 0$, where $\mathbf{L}(s)$ is an operator or a matrix that depends on the

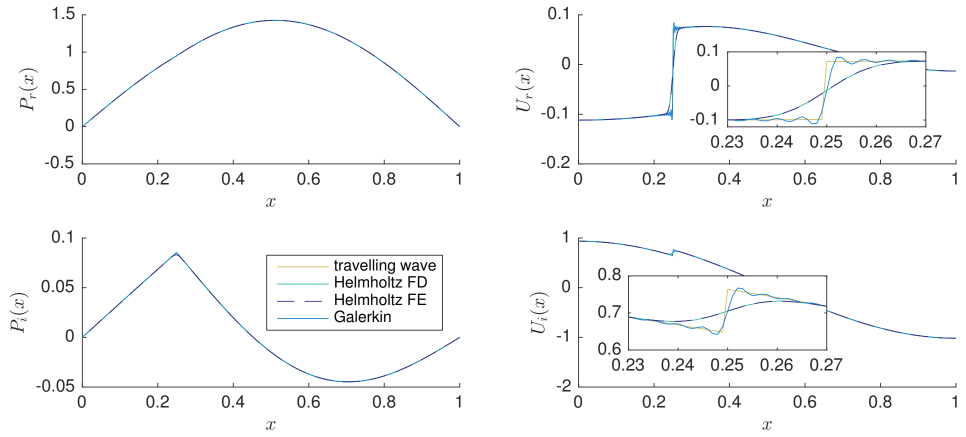

Figure 2

Real and imaginary components of the pressure, $P(x)$, and velocity, $U(x)$, eigenfunctions calculated for the highest resolution cases of the methods shown in table 1. The eigenfunctions lie on top of each other except in the heat release zone. In the heat release zone, the wave eigenfunction is discontinuous in u, the Galerkin eigenfunction exhibits Gibbs fringes in U, and the Helmholtz eigenfunction is smooth because the heat release is distributed over a region of space rather than at a point. (`Tab_comparisons.m`)

eigenvalue s, with corresponding eigenvector $\mathbf{q}$. The adjoint problem is constructed by pre-multiplying this expression by another function $\mathbf{q}^\dagger$:

$$\mathbf{q}^{\dagger H}\mathbf{L}(s)\mathbf{q} = 0\ . \tag{9}$$

For a given eigenvalue, s, the right eigenvector, $\mathbf{q}$, is defined such that (9) is satisfied for arbitrary $\mathbf{q}^\dagger$. Similarly, the left eigenvector, $\mathbf{q}^\dagger$, is defined such that (9) is satisfied for arbitrary $\mathbf{q}$. A change in $\mathbf{L}$ at order ϵ induces changes to s, $\mathbf{q}$, and $\mathbf{q}^\dagger$ at order ϵ. The eigenvalue drift, δs, is found by substituting these changes into $\mathbf{L}(s)\mathbf{q} = 0$ and pre-multiplying by $\mathbf{q}^{\dagger H}$. At order ϵ this gives:

$$\mathbf{q}^{\dagger H}\left(\mathbf{L}\delta\mathbf{q} + (\partial\mathbf{L}/\partial s)\delta s\mathbf{q} + \delta\mathbf{L}\mathbf{q}\right) = 0 \tag{10}$$

Noting that $\mathbf{q}^{\dagger H}\mathbf{L}\delta\mathbf{q} = 0$ for arbitrary $\delta\mathbf{q}$, this can be re-arranged to give the sensitivity of the eigenvalue, s, to a generic change in the operator, $\mathbf{L}(s)$:

$$\delta s = -\frac{\mathbf{q}^{\dagger H}(\delta\mathbf{L})\mathbf{q}}{\mathbf{q}^{\dagger H}(\partial\mathbf{L}/\partial s)\mathbf{q}} \tag{11}$$

Magri et al. (2016) provide full details and also show how to handle degenerate eigenvalues.

The base state sensitivity is the sensitivity of the eigenvalue to changes in the model parameters, which are (n, τ, x_m, x_f) in this case. All the numerical methods share the same model parameters but have different internal parameters. The gradients of s with respect to the model parameters are found with the chain rule. In the travelling wave method, for example, the internal parameters are $h, \tau, \tau_u, \tau_d, \tau_{mf}$ and the gradient of s with respect to the heat source position, x_f, is:

$$\frac{\partial s}{\partial x_f} = -\frac{\mathbf{q}^{\dagger H}(\partial\mathbf{L}/\partial x_f)\mathbf{q}}{\mathbf{q}^{\dagger H}(\partial\mathbf{L}/\partial s)\mathbf{q}} \quad \text{where} \quad \frac{\partial\mathbf{L}}{\partial x_f} = \frac{\mathrm{d}\tau_u}{\mathrm{d}x_f}\frac{\partial\mathbf{L}}{\partial\tau_u} + \frac{\mathrm{d}\tau_d}{\mathrm{d}x_f}\frac{\partial\mathbf{L}}{\partial\tau_d} + \frac{\mathrm{d}\tau_{mf}}{\mathrm{d}x_f}\frac{\partial\mathbf{L}}{\partial\tau_{mf}} \tag{12}$$

In the Helmholtz Finite Difference method, for example, the internal parameters are $(\mathbf{F}, \tau)$ and the code to calculate the sensitivities of s with respect to the model parameters is:

```
%% Calculate the gradients of the internal parameters w.r.t. param.*
% dv/d(param.x_f)
dvdx_f = (2*(x-param.x_f)/param.an^2).*v;
% dw/d(param.x_m)
dwdx_m = (2*(x-param.x_m)/param.an^2).*w;
% dF/d(param.n)
dFdn   = F / param.n;
% dF/d(param.x_f)
dFdx_f = (dvdx_f*w') * (param.gam-1)/param.gam * param.n * M * D;
% dF/d(param.x_m)
dFdx_m = (v*dwdx_m') * (param.gam-1)/param.gam * param.n * M * D;

%% Calculate the gradients of L w.r.t. to the internal parameters
% Calculate dL/ds
dLds = fun_dLds(I,F,t,s);
% Calcualte dL/dF
dLdF = fun_dLdF(t,s);
% Calculate dL/dt
dLdt = fun_dLdt(F,t,s);

%% Calculate the gradients of L w.r.t. param.*
dLdn   = dLdF * dFdn;
dLdx_m = dLdF * dFdx_m;
dLdx_f = dLdF * dFdx_f;

%% Calculate the normalizing inner product
nip = (P_adj' * dLds * P_dir);

%% Calculate the gradients of s w.r.t. param.*
ds.n   = - (P_adj' * dLdn   * P_dir) / nip;
ds.tau = - (P_adj' * dLdt   * P_dir) / nip;
ds.x_m = - (P_adj' * dLdx_m * P_dir) / nip;
ds.x_f = - (P_adj' * dLdx_f * P_dir) / nip;
```

Listing 9 Calculation of the base state sensitivities in `fun_Helm_FD.m`

The base state sensitivities calculated with each method are shown in Table 2. They match to the precisions expected given the small differences between the methods.

These base state sensitivities can also be calculated from $\delta s = s(\Pi + \epsilon) - s(\Pi)$, where Π is a parameter. The sensitivities calculated this way, labelled $(\delta s)_{FD}$, contain contributions at second and higher orders of ϵ, while the sensitivities calculated with the adjoint methods in this tutorial are exact to first order in ϵ. This permits an excellent debugging check for adjoint codes, known as a Taylor Test: the difference $|(\delta s)_{AD} - (\delta s)_{FD}|$ must increase in proportion to ϵ^2 or higher order. Conversely, if this difference increases in proportion to ϵ then there must be a bug in the adjoint code. This test can be performed for one parameter at a time or for all parameters simultaneously. Figure 3 plots $|(\delta s)_{AD} - (\delta s)_{FD}|$ against ϵ^2 when all base state parameters are changed simultaneously for all four methods. Each is a straight line through the origin, showing that the adjoint calculations are indeed exact to first order. It is worth introducing a small bug into the codes to see its influence on this test.

	$\partial s/\partial n$	$\partial s/\partial \tau$	$\partial s/\partial x_m$	$\partial s/\partial x_f$
`fun_travwave.m`	$+0.10298 + 0.08269i$	$+0.25395 - 0.34197i$	$-0.26649 - 0.17767i$	$+0.43129 + 0.21134i$
`fun_Helm_FD.m`	$+0.10294 + 0.08266i$	$+0.25384 - 0.34181i$	$-0.26635 - 0.17759i$	$+0.43104 + 0.21127i$
`fun_Helm_FE.m`	$+0.10294 + 0.08265i$	$+0.25384 - 0.34181i$	$-0.26635 - 0.17759i$	$+0.43104 + 0.21127i$
`fun_Galerkin.m`	$+0.10332 + 0.08257i$	$+0.25349 - 0.34304i$	$-0.26685 - 0.17750i$	$+0.39880 + 0.22590i$

$x_m = 0.20000,\ x_f = 0.25000,\ n = 1.00000,\ \tau = 0.34157,\ \gamma = 1.40000,\ R = -1.00000$

Table 2 Base state sensitivities calculated with the four methods with $N = 400$. (`Tab_base_state.m`)

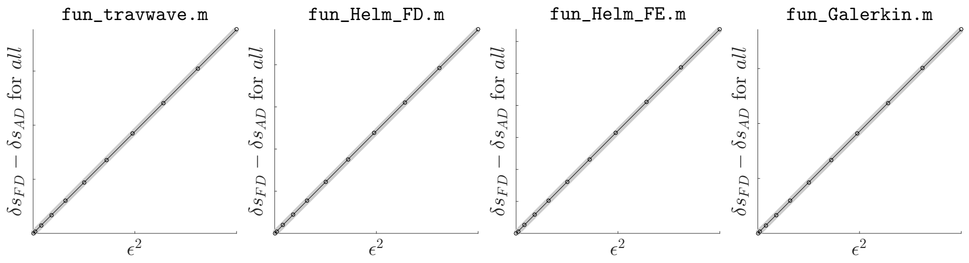

Figure 3

The difference between the eigenvalue drift calculated with (i) a finite difference method with step size ϵ (δs_{FD}) and (ii) adjoint methods (δs_{AD}). The adjoint methods are exact to first order so, for small ϵ, these plots should be straight lines through the origin. This provides an excellent debugging test for the adjoint codes, known as a Taylor Test. (`Fig_TT.m`)

LITERATURE CITED

Magri L, Bauerheim M, Juniper MP. 2016. Stability analysis of thermo-acoustic nonlinear eigenproblems in annular combustors. Part I Sensitivity. *Journal of Computational Physics* 325:395–410

Trefethen LN. 2000. Spectral Methods in Matlab. Philadelphia: SIAM

LIST OF MATLAB CODES PROVIDED WITH TUTORIAL

Main codes

`Tab_comparison.m`	Compare the eigenmodes calculated with the four methods by generating table 1 and figure 2.
`Tab_base_state.m`	Compare the sensitivities calculated with the four methods by generating table 2.
`Fig_TT.m`	Check that the adjoint sensitivities are exact to first order by generating figure 3.

Major functions

`fun_travwave.m`	Calculate the eigenmode and eigenvalue sensitivities with a travelling wave method.
`fun_Galerkin.m`	Calculate the eigenmode and eigenvalue sensitivities with a Galerkin method.
`fun_Helm_FD.m`	Calculate the eigenmode and eigenvalue sensitivities with a Helmholtz finite difference method.
`fun_Helf_FE.m`	Calculate the eigenmode and eigenvalue sensitivities with a Helmholtz finite element method.
`fun_TT.m`	Calculate and plot the difference between adjoint sensitivities and finite difference sensitivities.

Minor functions

`fun_param_dim.m`	Set the dimensional parameters of the thermoacoustic system.
`fun_nondim.m`	Convert to nondimensional parameters.
`fun_normalize.m`	Normalize an eigenvector, $P(x)$, such that $\int P^2 \,\mathrm{d}x = 1$.
`fun_cheb.m`	Compute a Chebyshev differentiation matrix and gridpoints (Trefethen 2000).
`fun_clencurt.m`	Compute the weights for a mass matrix with Clenshaw-Curtis quadrature (Trefethen 2000).