Notes on EVE: EigenVector Expansion Solver

Jeffrey W. Banks[a], William D. Henshaw[a]

[a]*Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, USA*

Abstract

We consider the numerical solution of initial-boundary-value problems IBVP's (or IVP's or Helmholtz problems) using an eigenfunction expansion approach. Discrete approximations to eigenfunctions for a given domain Ω are first computed. These eigefunctions are then used to represent the general solution. The solution at any time can then be found.

Contents

Email addresses: banksj3@rpi.edu (Jeffrey W. Banks), henshw@rpi.edu (William D. Henshaw)

1. Introduction

We consider the numerical solution of initial-boundary-value problems IBVP's (or IVP's or Helmholtz problems) using an eigenfunction expansion approach. Discrete approximations to eigenfunctions for a given domain Ω are first computed. These eigefunctions are then used to represent the general solution. The solution at any time can then be found.

1.1. EVE for the Wave Equation

Consider for example, the solution to the wave-like equation on a domain Ω, with some initial conditions and homogeneous boundary conditions,

$$\partial_t^2 u + Lu = 0, \qquad \mathbf{x} \in \Omega,\ t > 0 \tag{1a}$$
$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{1b}$$
$$\partial_t u(\mathbf{x}, 0) = u_1(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{1c}$$
$$\mathcal{B}u(\mathbf{x}, t) = 0 \qquad \mathbf{x} \in \partial\Omega, \tag{1d}$$

where $L = -c^2\Delta$, for example. We want to initially consider problems where there is a basis of ortho-normal eigenfunctions. Suppose we know the eigenvalues and eigenfunctions

$$L\phi_j = \lambda_j \phi_j, \quad \mathbf{x} \in \Omega \tag{2a}$$
$$\phi_j(\mathbf{x}) = 0 \qquad \mathbf{x} \in \partial\Omega, \tag{2b}$$

The solution be represented as an eigenfunction expansion,

$$u(\mathbf{x}, t) = \sum_{j=0}^{\infty} q_j(t)\phi_j(\mathbf{x}), \tag{3}$$

If the eigenfunctions are orthonormal, then

$$q_j'' = -\lambda_j q_j \tag{4}$$

and

$$\sum_{j=0}^{\infty} q_j(0)\phi_j(\mathbf{x}) = u_0(\mathbf{x}), \tag{5}$$
$$\sum_{j=0}^{\infty} q_j'(0)\phi_j(\mathbf{x}) = u_1(\mathbf{x}) \tag{6}$$

If we know the eigenfunction expansions for the initial conditions,

$$u_0(\mathbf{x}) = \sum_j \hat{u}_j \phi_j(\mathbf{x}), \tag{7}$$
$$u_1(\mathbf{x}) = \sum_j \hat{v}_j \phi_j(\mathbf{x}), \tag{8}$$

then

$$q_j(t) = \hat{u}_j \cos(\sqrt{\lambda_j}t) + \frac{1}{\sqrt{\lambda_j}}\hat{v}_j \sin(\sqrt{\lambda_j}t) \tag{9}$$

Thus the solution is

$$u(\mathbf{x},t) = \sum_j \left(\hat{u}_j \cos(\sqrt{\lambda_j}t) + \frac{1}{\sqrt{\lambda_j}}\hat{v}_j \sin(\sqrt{\lambda_j}t)\right)\phi_j(\mathbf{x}) \tag{10}$$

Note that if the initial conditions $u_0(\mathbf{x})$ and $u_1(\mathbf{x})$ are smooth and satisfy the boundary conditions then we expect their generalized Fourier series to converge rapidly (depending also on the smoothness of the domain).

1.2. Forced problem.

Note: in this section we use the new notation here: $L\phi_k = -\lambda_k^2\phi_k$, $L = c^2\Delta$.

Next consider the forced problem with homogeneous boundary conditions,

$$\partial_t^2 u + Lu = f(\mathbf{x},t), \quad \mathbf{x}\in\Omega,\ t>0 \tag{11a}$$
$$u(\mathbf{x},0) = u_0(\mathbf{x}), \quad \mathbf{x}\in\Omega, \tag{11b}$$
$$\partial_t u(\mathbf{x},0) = u_1(\mathbf{x}), \quad \mathbf{x}\in\Omega, \tag{11c}$$
$$\mathcal{B}u(\mathbf{x},t) = 0 \quad \mathbf{x}\in\partial\Omega, \tag{11d}$$

Let us suppose that $f(\mathbf{x},t)$ has a convergent eigenfunction expansion,

$$f(\mathbf{x}) = \sum_k \hat{f}_k t)\phi_k(\mathbf{x}). \tag{12}$$

If f is smooth and of compact support in Ω then this expansion should converge rapidly. If f is not zero at the boundary then the convergence may be slow near the boundaries (what should we do on this case?) As before, the solution be represented as an eigenfunction expansion,

$$u(\mathbf{x},t) = \sum_{k=0}^{\infty} q_k(t)\phi_k(\mathbf{x}), \tag{13}$$

where q_k satisfies the forced ODE IVP

$$q_k'' + \lambda_k^2 q_k = \hat{f}_k(t), \tag{14}$$
$$q_k(0) = \hat{u}_k, \tag{15}$$
$$q_k'(0) = \hat{v}_k \tag{16}$$

The general solution is finish me

$$q_k(t) = c_1\cos(\lambda_k t) + c_2\sin(\lambda_k t) + \int_0^t \hat{f}_k(\tau)\,\frac{\sin(\lambda_k(t-\tau))}{\lambda_k}\,d\tau, \tag{17}$$

When $f(\mathbf{x},t)$ is non-zero near the boundary (for Dirichlet BC's) then its Fourier series will only

converge in the mean. Suppose $\hat{f}_k \sim 1/k$ then the forcing term in (17) will be $O(1/k^3)$ if $\lambda_k \sim k$, I think, and the Fourier series for u will converge reasonably fast. Compare this to Section 1.4 on forced BCs which has slower convergence.

Depending on the form of $\hat{f}_k(t)$ the particular solution can be computed analytically, otherwise some quadrature rule can be used. These computations of q_k are easy to parallelize.

1.3. Inhomogeneous boundary conditions (I)

Next consider the case of inhomogeneous boundary conditions,

$$\partial_t^2 u + Lu = 0, \qquad \mathbf{x} \in \Omega,\ t > 0 \tag{18a}$$
$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{18b}$$
$$\partial_t u(\mathbf{x}, 0) = u_1(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{18c}$$
$$\mathcal{B}u(\mathbf{x}, t) = g(\mathbf{x}, t) \quad \mathbf{x} \in \partial\Omega, \tag{18d}$$

Suppose we knew any smooth function $\psi(\mathbf{x}, t)$ that satisfies the boundary conditions,

$$\psi(\mathbf{x}, t) = g(\mathbf{x}, t) \quad \mathbf{x} \in \partial\Omega, \tag{19}$$

Then we write

$$u(\mathbf{x}, t) = \psi(\mathbf{x}, t) + w(\mathbf{x}, t), \tag{20}$$

and we now solve for w which satisfies the forced equations with homogeneous boundary conditions

$$\partial_t^2 w + Lw = (\partial_t^2 + L)\psi, \qquad \mathbf{x} \in \Omega,\ t > 0 \tag{21a}$$
$$w(\mathbf{x}, 0) = u_0(\mathbf{x}) - \psi(\mathbf{x}, 0), \qquad \mathbf{x} \in \Omega, \tag{21b}$$
$$\partial_t w(\mathbf{x}, 0) = u_1(\mathbf{x}) - \partial_t \psi(\mathbf{x}, 0), \quad \mathbf{x} \in \Omega, \tag{21c}$$
$$\mathcal{B}w(\mathbf{x}, t) = 0 \qquad \mathbf{x} \in \partial\Omega, \tag{21d}$$

The problem has thus been reduced to one with homogeneous boundary conditions. There is still the issue of representing the initial conditions and forcing as a generalized Fourier series.

There are some special cases. If $g = g(\mathbf{x})$ is time independent we could choose ψ to solve the BVP

$$L\psi = 0, \qquad \mathbf{x} \in \Omega \tag{22a}$$
$$\mathcal{B}\psi = g(\mathbf{x}) \quad \mathbf{x} \in \partial\Omega, \tag{22b}$$

which removes the forcing term from the equation for w (but does alter the initial conditions).

1.4. Inhomogeneous boundary conditions (II)

Note: in this section we use the new notation here: $L\phi_k = -\lambda_k^2 \phi_k$, $L = c^2\Delta$.

There is another way to treat inhomogeneous boundary conditions that does not require the construction of function that satisfies the boundary conditions. Consider

$$\partial_t^2 u + c^2 \Delta u = 0, \qquad \mathbf{x} \in \Omega,\ t > 0 \tag{23a}$$

$$u(\mathbf{x},0) = u_0(\mathbf{x}), \qquad \mathbf{x}\in\Omega, \tag{23b}$$
$$\partial_t u(\mathbf{x},0) = u_1(\mathbf{x}), \quad \mathbf{x}\in\Omega, \tag{23c}$$
$$\mathcal{B}u(\mathbf{x},t) = g(\mathbf{x},t) \quad \mathbf{x}\in\partial\Omega, \tag{23d}$$

The EVE solution is of the form

$$u(\mathbf{x},t)\sum_k \hat{u}_k(t)\phi_k(\mathbf{x}), \tag{24}$$
$$\hat{u}_k(t) = \int_\Omega u(\mathbf{x},t)\phi_k(\mathbf{x})\,d\mathbf{x}. \tag{25}$$

Some care is required when the solution is written as (24) since the convergence is only in the mean near the boundary. All eigenfunctions are zero at the boundary so $\mathbf{u}$ is also zero at the boundary. However we can start from (25) and take two time-derivatives,

$$\partial_t^2\hat{u}_k(t) = \int_\Omega \partial_t^2 u(\mathbf{x},t)\phi_k(\mathbf{x})\,d\mathbf{x}, \tag{26}$$
$$= \int_\Omega c^2\nabla\cdot(\nabla u)\,\phi_k(\mathbf{x})\,d\mathbf{x}. \tag{27}$$

Now integrating by parts, check me

$$\partial_t^2\hat{u}_k(t) = \int_{\partial\Omega} c^2\partial_n u\,\phi_k\,dS - \int_\Omega c^2\nabla u\cdot\nabla\phi_k(\mathbf{x})\,d\mathbf{x}, \tag{28}$$
$$= \int_{\partial\Omega} c^2\partial_n u\,\phi_k\,dS - \int_{\partial\Omega} c^2 u\,\partial_n\phi_k\,dS + \int_\Omega u\,c^2\Delta\phi_k(\mathbf{x})\,d\mathbf{x}. \tag{29}$$

Using the boundary condition and that fact that $\phi_k(\mathbf{x})$ is zero on the boundary leads to

$$\partial_t^2\hat{u}_k(t) = -\int_{\partial\Omega} c^2 g(\mathbf{x},t)\,\partial_n\phi_k\,dS + \lambda_k^2\int_\Omega u(\mathbf{x},t)\,\phi_k(\mathbf{x})\,d\mathbf{x}, \tag{30}$$

or

$$\partial_t^2\hat{u}_k(t) + \lambda_k^2\hat{u}_k = -\int_{\partial\Omega} c^2 g(\mathbf{x},t)\,\partial_n\phi_k\,dS. \tag{31}$$

Equation (31) defines the ODE for the Fourier coefficients that directly incorporates the inhomogeneous BCs.

1.4.1. Example: inhomogeneous boundary conditions

Note: in this section we use the new notation here: $L\phi_k = -\lambda_k^2\phi_k$, $L = c^2\Delta$.

Consider the 1D IBVP with homogeneous initial conditions and a non-homogeneous BC at $x = 0$.

$$\partial_t^2 u + c^2\partial_x^2 = 0, \quad x\in(0,\pi),\ t>0 \tag{32a}$$
$$u(x,0) = 0, \qquad x\in(0,\pi), \tag{32b}$$
$$\partial_t u(x,0) = 0, \qquad x\in(0,\pi), \tag{32c}$$

$$u(0,t) = g(t), \qquad t > 0, \tag{32d}$$
$$u(1,t) = 0, \qquad t > 0. \tag{32e}$$

The eigenvalues and orthonormal eigenfunctions are

$$\lambda_k = ck, \quad \phi_k(x) = \sqrt{\frac{2}{\pi}}\sin(kx), \qquad k = 1, 2, \ldots. \tag{33}$$

Equation (31) for the Fourier coefficients becomes

$$\partial_t^2 \hat{u}_k(t) + \lambda_k^2 \hat{u}_k = c^2 g(t)\, \partial_x \phi_k(0) = c^2 k \sqrt{\frac{2}{\pi}} g(t) \stackrel{\text{def}}{=} G_k(t). \tag{34}$$

Note that the RHS is proportional to k which implies the convergence of the series may not be so fast. The general solution is

$$\hat{u}_k(t) = c_1 \cos(\lambda_k t) + c_2 \sin(\lambda_k t) + \int_0^t G_k(\tau)\, \frac{\sin(\lambda_k(t-\tau))}{\lambda_k}\, d\tau, \tag{35}$$

Imposing the initial conditions implies check me

$$\hat{u}_k(t) = \int_0^t G_k(\tau)\, \frac{\sin(\lambda_k(t-\tau))}{\lambda_k}\, d\tau. \tag{36}$$

Example 1:

$$g(t) = \sin(\alpha t). \tag{37}$$

Then (for $\alpha \neq ck$) check me

$$\hat{u}_k(t) = \sqrt{\frac{2}{\pi}}\, \frac{c^2}{c^2k^2 - \alpha^2} \Big[k \sin(\alpha t) - \alpha \sin(ckt) \Big] \tag{38}$$

If $\alpha = cm$ for an m then $\hat{u}_m$ is forced at resonance and the form of the solution differs.

Example 2: (smoother start)

$$g(t) = \frac{1}{2}(1 - \cos(\alpha t)) = \sin^2(\alpha t/2). \tag{39}$$

Then (for $\alpha \neq ck$) check me

$$\hat{u}_k(t) = \frac{1}{2}\sqrt{\frac{2}{\pi}} c^2 k \Big[\frac{1}{c^2k^2} - \frac{1}{c^2k^2 - \alpha^2} \cos(\alpha t) - \big(\frac{1}{c^2k^2} - \frac{1}{c^2k^2 - \alpha^2} \big) \cos(kt) \Big]. \tag{40}$$

Example 3: (impulsive start)

$$g(t) = 1. \tag{41}$$

Then

$$\hat{u}_k(t) = \sqrt{\frac{2}{\pi}}\frac{1}{k}\big(1-\cos(ckt)\big). \tag{42}$$

The exact solution is (until the wave hits the right side)

$$u(x,t) = \begin{cases} g(\alpha(t-x/c)) & x \leqslant ct, \\ 0 & x > ct \end{cases} \tag{43}$$

Figure 1 shows some results. The convergence is slow near $x = 0$. NOTE. We should be able to post-process the solution to get a better answer – see Section 1.6.

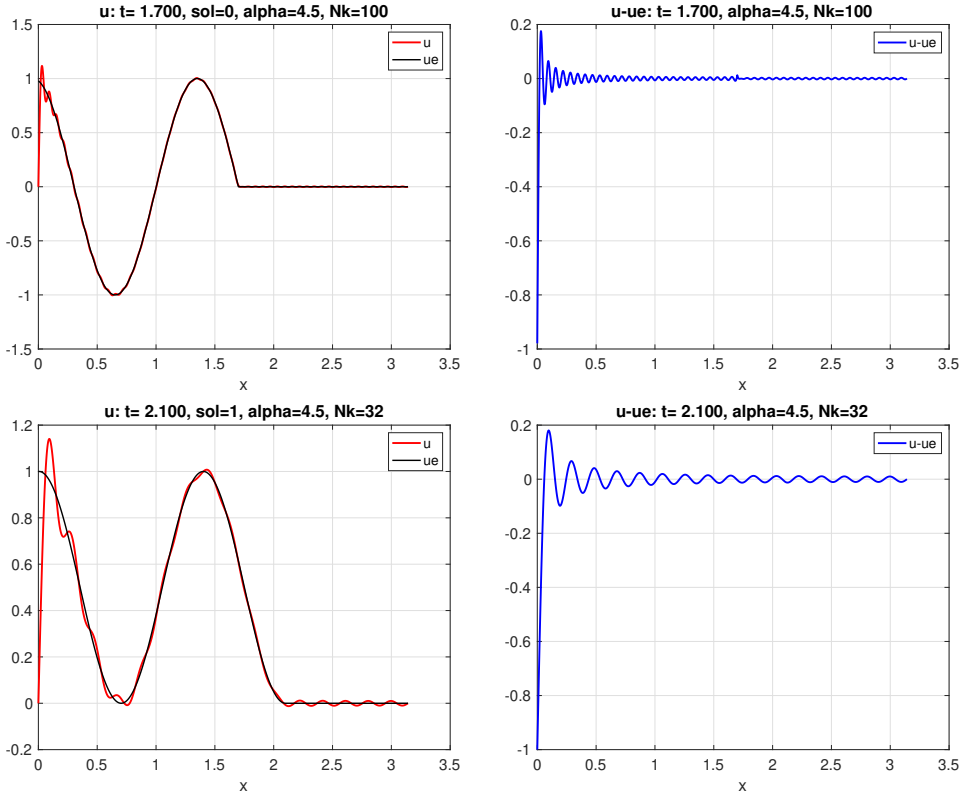

Figure 1: Inhomogeneous BC. Top: boundary forcing $g(t) = \sin(\alpha t)$. Bottom: $g(t) = \frac{1}{2}(1-\cos(\alpha t)) = \sin^2(\alpha t/2)$.

1.5. Correcting the Fourier series

Figures 2 and 3 show the corrected solutions and errors using the correction function described in 1.6.2. The corrected solution is

$$u_c(x) = S_N u - g(t)(S_N v - v), \tag{44}$$

$$v \stackrel{\text{def}}{=} 1 - x/\pi. \tag{45}$$

Here $S_N u$ denotes the Fourier series with N terms The corrected solutions appear to be converging at second-order.

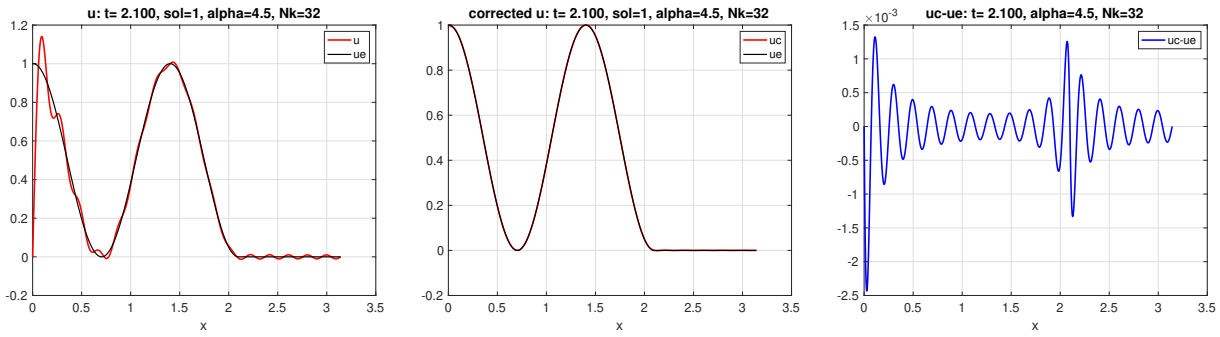

Figure 2: Inhomogeneous BC with correction, $N_k = 32$. $g(t) = \frac{1}{2}(1 - \cos(\alpha t)) = \sin^2(\alpha t/2)$.

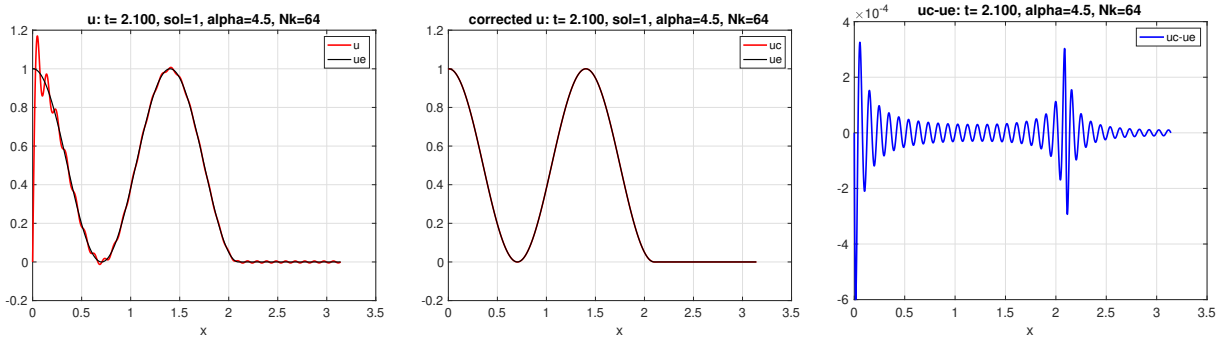

Figure 3: Inhomogeneous BC with correction, $N_k = 64$. $g(t) = \frac{1}{2}(1 - \cos(\alpha t)) = \sin^2(\alpha t/2)$.

1.5.1. Error analysis of the corrected solution

The error in the corrected solution u_c

$$u_c(x,t) = S_N u - g(t)(S_N v - v), \tag{46}$$

is

$$\begin{aligned} e_N(x,t) &= u - u_c && (47)\\ &= (u - g(t)v) - S_N(u - g(t)v) && (48)\\ &= w - S_N w && (49) \end{aligned}$$

where

$$w \stackrel{\text{def}}{=} u - g(t)v. \tag{50}$$

The error e_N is thus the error in the N-th partial Fourier sum of w. If $w(x,t)$ is smooth enough then the error will go to zero point-wise, and the convergence rate will depend on how smooth w is.

Let's now show the conditions that lead to a smooth $w(x,t)$. It follows that w satisfies the PDE

$$\begin{aligned} \partial_t^2 w - Lw &= (\partial_t^2 u - Lu) - (\partial_t^2 g(t)v(x) - g(t)Lv), && (51)\\ &= -(\partial_t^2 g(t)v(x) - g(t)Lv), && (52)\\ &= -\partial_t^2 g(t)\, v(x) && (53) \end{aligned}$$

using $Lv = 0$ (all we probably need is that Lv is smooth enough), and by construction w has homogeneous boundary conditions

$$w(x,t) = 0, \qquad x = 0, \pi \tag{54}$$

and the initial conditions are

$$w(x,0) = u(x,0) - g(0)v(x), \qquad \partial_t w(x,0) = \partial_t u(x,0) - \partial_t g(0)v(x). \tag{55}$$

To summarize, $w(x,t)$ satisfies the forced IBVP with homogeneous BCs

$$\begin{aligned} &\partial_t^2 w - Lw = -\partial_t^2 g(t)\, v(x), && (56a)\\ &w(x,0) = u(x,0) - g(0)v(x), \qquad \partial_t w(x,0) = \partial_t u(x,0) - \partial_t g(0)v(x), && (56b)\\ &w(x,t) = 0, \qquad x = 0, \pi && (56c) \end{aligned}$$

It should follow that $w(x,t)$ is smooth enough so that it's Fourier series converges pointwise. Note that if If u has homogeneous initial conditions, $u(x,t)$ and $\partial_t u(x,0) = 0$ (e.g. if we have decomposed the problem into one with homogeneous initial conditions and inhomogeneous BCs) and $g(0) = g'(0) = 0$ then w will have homogeneous initial conditions.

Note. This shows that following two approaches are equivalent

1. Subtract off a smooth function $G(x,t)$ that satisfies the BCs, solve the forced IBVP (56) for the correction, and then add back $G(x,t)$.
2. Solve the uncorrected IBVP to some time t (using the integration-by-parts trick to define the ODEs for $\hat{u}_k$) and then correct the solution.

The second method has benefits since we do not need to find $G(x,t)$ for all times (which may be expensive in more general geometries where $G(x,t)$ is not of the form $G_1(x)G_2(t)$).

1.5.2. Correcting the Fourier series in a general geometry

Consider a more general IBVP in n_d space dimensions

$$\begin{aligned} &\partial_t^2 u = Lu, && \mathbf{x} \in \Omega, && (57)\\ &u(\mathbf{x},t) = \partial_t u(\mathbf{x},0) = 0, && \mathbf{x} \in \Omega, && (58) \end{aligned}$$

$$u(\mathbf{x},t) = g(\mathbf{x},t) \qquad \mathbf{x} \in \partial\Omega, \tag{59}$$

which we solve with an eigenfunction expansion ($L\phi_k = -\lambda_k^2 \phi_k$)

$$S_N u = \sum_{k=1}^{N} \hat{u}_k(t)\phi_k(\mathbf{x}), \tag{60}$$

$$\hat{u}_k = \int_\Omega \phi_k(\mathbf{x})\, u(\mathbf{x},t)\, dx, \tag{61}$$

using the integration-by-parts trick to define the ODEs for $\hat{u}_k$.

Let $v(\mathbf{x},t)$ solve the elliptic BVP (this is not really required – we just need v to satisfy the BCs and Lv smooth enough)

$$Lv = 0, \qquad \mathbf{x} \in \Omega, \tag{62}$$

$$v(\mathbf{x},t) = g(\mathbf{x},t) \qquad \mathbf{x} \in \partial\Omega, \tag{63}$$

The corrected solution is then

$$u_c(\mathbf{x},t) = S_N u + (v - S_N v). \tag{64}$$

The error in the corrected solution u_c

$$e_N(\mathbf{x},t) = u - u_c \tag{65}$$

$$= (u - v) - S_N(u - v(\mathbf{x},t)) \tag{66}$$

$$= w - S_N w, \tag{67}$$

where

$$w(\mathbf{x},t) \stackrel{\text{def}}{=} u(\mathbf{x},t) - v(\mathbf{x},t). \tag{68}$$

The error e_N is thus the error in the N-th partial Fourier sum of w. If $w(\mathbf{x},t)$ is smooth enough then the error will go to zero point-wise, and the convergence rate will depend on how smooth w is.

Proceeding as before, it follows that $w(x,t)$ satisfies the forced IBVP with homogeneous BCs

$$\partial_t^2 w - Lw = -(\partial_t^2 v - Lv), \qquad \mathbf{x} \in \Omega, \tag{69a}$$

$$w(\mathbf{x},0) = u(\mathbf{x},0) - v(\mathbf{x},0) \stackrel{\text{def}}{=} w_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{69b}$$

$$\partial_t w(\mathbf{x},0) = \partial_t u(\mathbf{x},0) - \partial_t v(\mathbf{x},0) \stackrel{\text{def}}{=} w_1(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{69c}$$

$$w(\mathbf{x},t) = 0, \qquad \mathbf{x} \in \partial\Omega, \tag{69d}$$

where for the choice of v from above, $Lv = 0$, but this is not really required. If v is smooth enough, then w will be smooth enough and the generalized Fourier series for w should converge point-wise. This implies the corrected solution will also converge point-wise.

The convergence of the Fourier series (FS) for w will be primarily limited by the convergence of the FS for G defined by (the initial conditions $w_0(\mathbf{x})$ and $w_1(\mathbf{x})$ will also be a factor but note

that these functions are zero on the boundary)

$$G \stackrel{\text{def}}{=} \partial_t^2 v - Lv \tag{70}$$

in (69). For example, if $G(\mathbf{x}) \neq 0$ on the boundary then the FS for G will haves Gibbs effects near the boundary (but also note that the FS for w converges faster than the FS for G). If we want faster convergence at the boundary we could choose v so that G is zero on the boundary,

$$G(\mathbf{x}) = 0, \qquad \mathbf{x} \in \partial\Omega \tag{71}$$

which implies

$$Lv = \partial_t^2 v = \partial_t^2 g(\mathbf{x}, t) \qquad \mathbf{x} \in \partial\Omega \tag{72}$$

Question: Can we construct a smooth function local to each boundary and then combine these to form a composite v that satisfies all the boundary conditions and other constraints we want such as (72).

1.6. Filtering the Fourier series solution

1.6.1. Fourier series for a Heaviside function

The errors in Figure 1 look very much like the finite Fourier series for a step function. Figure 4 shows a Fourier series solution for a periodic top-hat on the domain $\Omega = [-\pi, \pi]$.

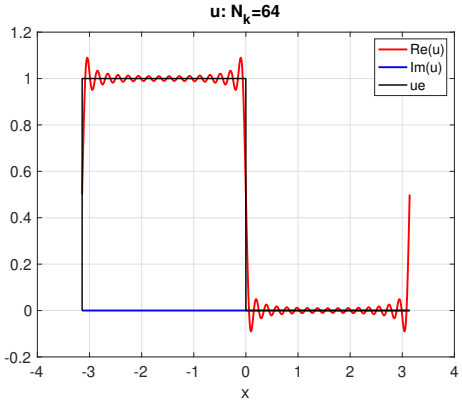

Figure 4: Fourier series for a top hat on $x \in [-\pi, \pi]$ showing the Gibbs phenomena.

1.6.2. Fourier series for $1 - x/\pi$

Mu guess is that the errors in Figure 1 should likely be mainly due to the errors in the Fourier sine series for the linear function

$$v(x) = 1 - x/\pi, \qquad x \in [0, \pi) \tag{73}$$

This function is 1 at $x = 0$ and 0 at $x = \pi$. The Fourier sine series is

$$S_N(v) = \sum_{k=1}^{N} \hat{v}_k \sin(kx), \tag{74}$$

$$\hat{v}_k = \frac{2}{\pi} \int_0^{\pi} v(x) \sin(kx)\, dx. \tag{75}$$

Using

$$\int_0^{\pi} \sin(kx) dx = \frac{-1}{k} \cos(kx)\Big|_0^{\pi} = \frac{1}{k}(1 - \cos(k\pi)), \tag{76}$$

$$\int_0^{\pi} x \sin(kx) dx = \Big[-1kx \cos(kx) + \frac{1}{k^2} \sin(kx) \Big]_0^{\pi} = \frac{-1}{k} \cos(k\pi), \tag{77}$$

gives

$$\hat{v}_k = \frac{2}{k\pi} \tag{78}$$

Figure 5 shows the series and error.

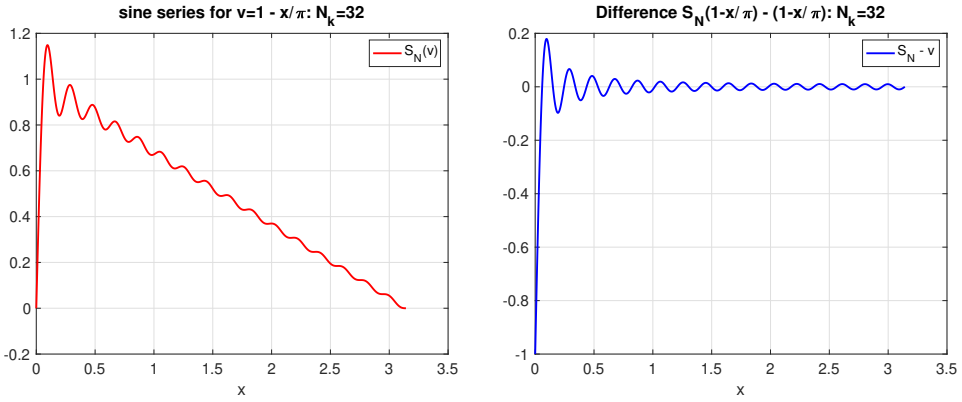

Figure 5: Fourier series for $v = 1 - x/\pi$ and the error.

2. Parallel in time solver

Suppose that we need to compute the solution to the IBVP quickly and we have a big parallel computer with lots of memory and lots of processors.

A typical finite-difference or finite element solver computes the solution to the IBVP at some t by time-marching from time zero. The solver can be parallelized in space but there is no obvious way to parallelize this approach in time due to the sequential nature of the time-stepping algorithm. Thus we limited in our parallel speedup. Time-stepping thus becomes the sequential bottleneck.

The eigenfunction expansion (10), however, can be used to evaluate the solution at any arbitrary time. This expansion can be used as the basis for a parallel in time solver. Once we know the eigenfunctions $\phi_j(\mathbf{x})$ then the solution at any time t can be often be computed at a fixed cost, independent of the time t, by summing the expansion for a finite number of terms (to the desired accuracy).

If the eigenfunctions are not known, they can be computed, in parallel. Different eigenfunctions can be computed independently if needed.

3. Motivational examples

What are some good motivational examples that can make use of the method?

1. Acoustics: Given a structure (or under-water vehicle,say), compute the maximum pressure on the structure for pressure sources located at different locations around the structure. Structure is fixed but solutions to many initial value problems are desired.
2. Electromagnetics : inverse design of the dispersive properties of meta-atoms in a dispersive material. Geometry of meta-atoms is fixed but dispersive properties can vary. Far field response for a broad-band signal is desired.
3. Elasticity: similar problem to acoustics – find maximum stress on a elastic structure due to different source terms.
4. Seismic applications :

4. Computing eigenfunctions and fitting functions

The routine `genEigs.bC` and class `Ogev` can be used to compute discrete approximations to eigenfunctions on overset grids using the `SLEPc` solver.

Notes:

1. SLEPc does not generally return orthogonal eigenvectors for the eigenspace of a multiple eigenvalue.
2. The eigenvectors for a multiple eigenvalue can be orthogonalized (e.g. using a QR algorithm). However, it appears that sometimes SLEPc returns essentially the same eigenvector more than once for a multiple eigenvalue. In this case the orthogonalization fails – need to fix this. May be able to provide SLEPc with some starting guesses.

4.1. Timings to compute eigenvalues and eigenvectors

Here are some timings for computing the eigenvalues/eigenvectors of the RPI grid (serial, cg6).

```
genEigs -noplot eigs.cmd -problem=laplace -eigCase=rpi -g=rpiGride4pn.order2.hdf -eigOption=1
-eps_s=largest_magnitude -numEigenValues=260 -numEigenVectors=256 -tol=1.e-12 -bc5=n -bc6=n -bc7=n

--------------- Ogev : computeEigenvalues --------------------
   numberOfGridPoints=81975, numEigenvalues=260, numEigenVectors=256
      total cpu =  1.04e+02 (s) (includes build matrix)
 cpu/eigenvalue =  4.02e-01 (s)
 Number of iterations of the method: 2
 Number of linear iterations of the method: 665
 Solution method: krylovschur


genEigs -noplot eigs.cmd -problem=laplace -eigCase=rpi -g=rpiGride4pn.order2.hdf -eigOption=1
-eps_s=largest_magnitude -numEigenValues=130 -numEigenVectors=128 -tol=1.e-12 -bc5=n -bc6=n -bc7=n

--------------- Ogev : computeEigenvalues --------------------
   numberOfGridPoints=81975, numEigenvalues=130, numEigenVectors=128
      total cpu =  3.62e+01 (s) (includes build matrix)
 cpu/eigenvalue =  2.78e-01 (s)
 Number of iterations of the method: 2
 Number of linear iterations of the method: 337
 Solution method: krylovschur


genEigs -noplot eigs.cmd -problem=laplace -eigCase=rpi -g=rpiGride4pn.order2.hdf -eigOption=1
 -eps_s=largest_magnitude -numEigenValues=20 -numEigenVectors=16 -tol=1.e-12 -bc5=n -bc6=n -bc7=n

--------------- Ogev : computeEigenvalues --------------------
   numberOfGridPoints=81975, numEigenvalues=20, numEigenVectors=16
      total cpu =  1.24e+01 (s) (includes build matrix)
 cpu/eigenvalue =  6.20e-01 (s)
 Number of iterations of the method: 3
 Number of linear iterations of the method: 67
 Solution method: krylovschur
```

4.2. Computed eigenfunctions of a square

Here are some computed eigenvalues for a square. It looks like high-order accurate schemes are useful if a high degree of accuracy is needed. Even double eigenvalues can be computed accurately in this case.

ORDER=4

```
genEigs -noplot eigs.cmd -problem=laplace -g=square64.order4 -eigOption=1 -eps_s=largest_magnitude
-numEigenValues=70  -numEigenVectors=64 -tol=1.e-12 -bc1=d -show=squareEigs64.show
Eigenvalue   0 : k=   19.7392082729 + (-0) I,  true=1.97392088021787e+01, err=5.29e-07, rel-err=2.68e-08
Eigenvalue   1 : k=   49.3480045834 + (-0) I,  true=4.93480220054468e+01, err=1.74e-05, rel-err=3.53e-07
Eigenvalue   2 : k=   49.3480045834 + (-0) I,  true=4.93480220054468e+01, err=1.74e-05, rel-err=3.53e-07
Eigenvalue   3 : k=   78.9568008939 + (-0) I,  true=7.89568352087149e+01, err=3.43e-05, rel-err=4.35e-07
Eigenvalue   4 : k=   98.6958441746 + (-0) I,  true=9.86960440108936e+01, err=2.00e-04, rel-err=2.02e-06
Eigenvalue   5 : k=   98.6958441746 + (-0) I,  true=9.86960440108936e+01, err=2.00e-04, rel-err=2.02e-06
```

```
Eigenvalue   6 : k=  128.3046404852 + (-0) I,  true=1.28304857214162e+02, err=2.17e-04, rel-err=1.69e-06
Eigenvalue   7 : k=  128.3046404852 + (-0) I,  true=1.28304857214162e+02, err=2.17e-04, rel-err=1.69e-06
Eigenvalue   8 : k=  167.7821211200 + (-0) I,  true=1.67783274818519e+02, err=1.15e-03, rel-err=6.88e-06
Eigenvalue   9 : k=  167.7821211200 + (-0) I,  true=1.67783274818519e+02, err=1.15e-03, rel-err=6.88e-06
Eigenvalue  10 : k=  177.6524800764 + (-0) I,  true=1.77652879219608e+02, err=3.99e-04, rel-err=2.25e-06
Eigenvalue  11 : k=  197.3909174305 + (-0) I,  true=1.97392088021787e+02, err=1.17e-03, rel-err=5.93e-06
Eigenvalue  12 : k=  197.3909174305 + (-0) I,  true=1.97392088021787e+02, err=1.17e-03, rel-err=5.93e-06
Eigenvalue  13 : k=  246.7387570217 + (-0) I,  true=2.46740110027234e+02, err=1.35e-03, rel-err=5.48e-06
Eigenvalue  14 : k=  246.7387570217 + (-0) I,  true=2.46740110027234e+02, err=1.35e-03, rel-err=5.48e-06
Eigenvalue  15 : k=  256.6051595574 + (-0) I,  true=2.56609714428323e+02, err=4.55e-03, rel-err=1.78e-05
```

ORDER=4 : Finer grid. Note quite 4th-order convergence. *check me*

```
 genEigs -noplot eigs.cmd -problem=laplace -g=square128.order4 -eigOption=1 -eps_s=largest_magnitude
  -numEigenValues=70  -numEigenVectors=64 -tol=1.e-14 -bc1=d

Eigenvalue   0 : k=   19.7392087459 + (-0) I,  true=1.97392088021787e+01, err=5.63e-08, rel-err=2.85e-09
Eigenvalue   1 : k=   49.3480201749 + (-0) I,  true=4.93480220054468e+01, err=1.83e-06, rel-err=3.71e-08
Eigenvalue   2 : k=   49.3480201749 + (-0) I,  true=4.93480220054468e+01, err=1.83e-06, rel-err=3.71e-08
Eigenvalue   3 : k=   78.9568316039 + (-0) I,  true=7.89568352087149e+01, err=3.60e-06, rel-err=4.57e-08
Eigenvalue   4 : k=   98.6960234232 + (-0) I,  true=9.86960440108936e+01, err=2.06e-05, rel-err=2.09e-07
Eigenvalue   5 : k=   98.6960234232 + (-0) I,  true=9.86960440108936e+01, err=2.06e-05, rel-err=2.09e-07
Eigenvalue   6 : k=  128.3048348522 + (-0) I,  true=1.28304857214162e+02, err=2.24e-05, rel-err=1.74e-07
Eigenvalue   7 : k=  128.3048348522 + (-0) I,  true=1.28304857214162e+02, err=2.24e-05, rel-err=1.74e-07
Eigenvalue   8 : k=  167.7831590477 + (-0) I,  true=1.67783274818519e+02, err=1.16e-04, rel-err=6.90e-07
Eigenvalue   9 : k=  167.7831590477 + (-0) I,  true=1.67783274818519e+02, err=1.16e-04, rel-err=6.90e-07
Eigenvalue  10 : k=  177.6528381005 + (-0) I,  true=1.77652879219608e+02, err=4.11e-05, rel-err=2.31e-07
Eigenvalue  11 : k=  197.3919704767 + (-0) I,  true=1.97392088021787e+02, err=1.18e-04, rel-err=5.95e-07
Eigenvalue  12 : k=  197.3919704767 + (-0) I,  true=1.97392088021787e+02, err=1.18e-04, rel-err=5.95e-07
Eigenvalue  13 : k=  246.7399737250 + (-0) I,  true=2.46740110027234e+02, err=1.36e-04, rel-err=5.52e-07
Eigenvalue  14 : k=  246.7399737250 + (-0) I,  true=2.46740110027234e+02, err=1.36e-04, rel-err=5.52e-07
Eigenvalue  15 : k=  256.6092717758 + (-0) I,  true=2.56609714428323e+02, err=4.43e-04, rel-err=1.73e-06
```

ORDER=8 – this seems like a good way to get accurate eigenvalues

```
genEigs -noplot eigs.cmd -problem=laplace -g=square128.order8 -eigOption=1 -eps_s=largest_magnitude
  -numEigenValues=70  -numEigenVectors=64 -tol=1.e-14 -bc1=d

Eigenvalue   0 : k=   19.7392088022 + (-0) I,  true=1.97392088021787e+01, err=9.44e-12, rel-err=4.78e-13
Eigenvalue   1 : k=   49.3480220055 + (-0) I,  true=4.93480220054468e+01, err=6.97e-12, rel-err=1.41e-13
Eigenvalue   2 : k=   49.3480220055 + (-0) I,  true=4.93480220054468e+01, err=1.10e-11, rel-err=2.22e-13
Eigenvalue   3 : k=   78.9568352087 + (-0) I,  true=7.89568352087149e+01, err=1.43e-11, rel-err=1.81e-13
Eigenvalue   4 : k=   98.6960440110 + (-0) I,  true=9.86960440108936e+01, err=1.44e-10, rel-err=1.46e-12
Eigenvalue   5 : k=   98.6960440110 + (-0) I,  true=9.86960440108936e+01, err=1.55e-10, rel-err=1.57e-12
Eigenvalue   6 : k=  128.3048572143 + (-0) I,  true=1.28304857214162e+02, err=1.49e-10, rel-err=1.16e-12
Eigenvalue   7 : k=  128.3048572143 + (-0) I,  true=1.28304857214162e+02, err=1.54e-10, rel-err=1.20e-12
Eigenvalue   8 : k=  167.7832748210 + (-0) I,  true=1.67783274818519e+02, err=2.45e-09, rel-err=1.46e-11
Eigenvalue   9 : k=  167.7832748210 + (-0) I,  true=1.67783274818519e+02, err=2.45e-09, rel-err=1.46e-11
Eigenvalue  10 : k=  177.6528792199 + (-0) I,  true=1.77652879219608e+02, err=2.94e-10, rel-err=1.66e-12
Eigenvalue  11 : k=  197.3920880242 + (-0) I,  true=1.97392088021787e+02, err=2.45e-09, rel-err=1.24e-11
Eigenvalue  12 : k=  197.3920880242 + (-0) I,  true=1.97392088021787e+02, err=2.45e-09, rel-err=1.24e-11
```

Figure **??** shows some computed eigenfunctions of a square.

4.3. Computed eigenfunctions of a disk

There are many multiplicity-two eigenvalues for the disk dues to the angular symmetry; these correspond to eigenfunctions of the forms

$$\phi_1(r,\theta) = J_n(\kappa r)\cos(n\theta), \tag{79}$$
$$\phi_2(r,\theta) = J_n(\kappa r)\sin(n\theta), \tag{80}$$

Here are some computed eigenvalues for a disk.

```
genEigs -noplot eigs.cmd -problem=laplace -eigCase=disk -g=sice4.order4 -eigOption=1 -eps_s=largest_magnitude
 -numEigenValues=40 -numEigenVectors=32 -tol=1.e-12 -bc1=d
Eigenvalue   0 : k=    5.7831856376 + (-0) I,  true=5.78318596294677e+00, err=3.25e-07, rel-err=5.63e-08
Eigenvalue   1 : k=   14.6819663423 + (-0) I,  true=1.46819706421239e+01, err=4.30e-06, rel-err=2.93e-07
Eigenvalue   2 : k=   14.6819663423 + (-0) I,  true=1.46819706421239e+01, err=4.30e-06, rel-err=2.93e-07
Eigenvalue   3 : k=   26.3745765958 + (-0) I,  true=2.63746164271634e+01, err=3.98e-05, rel-err=1.51e-06
Eigenvalue   4 : k=   26.3746064090 + (-0) I,  true=2.63746164271634e+01, err=1.00e-05, rel-err=3.80e-07
Eigenvalue   5 : k=   30.4712325478 + (-0) I,  true=3.04712623436621e+01, err=2.98e-05, rel-err=9.78e-07
Eigenvalue   6 : k=   40.7063642227 + (-0) I,  true=4.07064658182003e+01, err=1.02e-04, rel-err=2.50e-06
Eigenvalue   7 : k=   40.7063642227 + (-0) I,  true=4.07064658182003e+01, err=1.02e-04, rel-err=2.50e-06
Eigenvalue   8 : k=   49.2183375312 + (-0) I,  true=4.92184563216946e+01, err=1.19e-04, rel-err=2.41e-06
```

Figure 6 shows some computed eigenfunctions of a disk, arranged by magnitude of the eigenvalues.

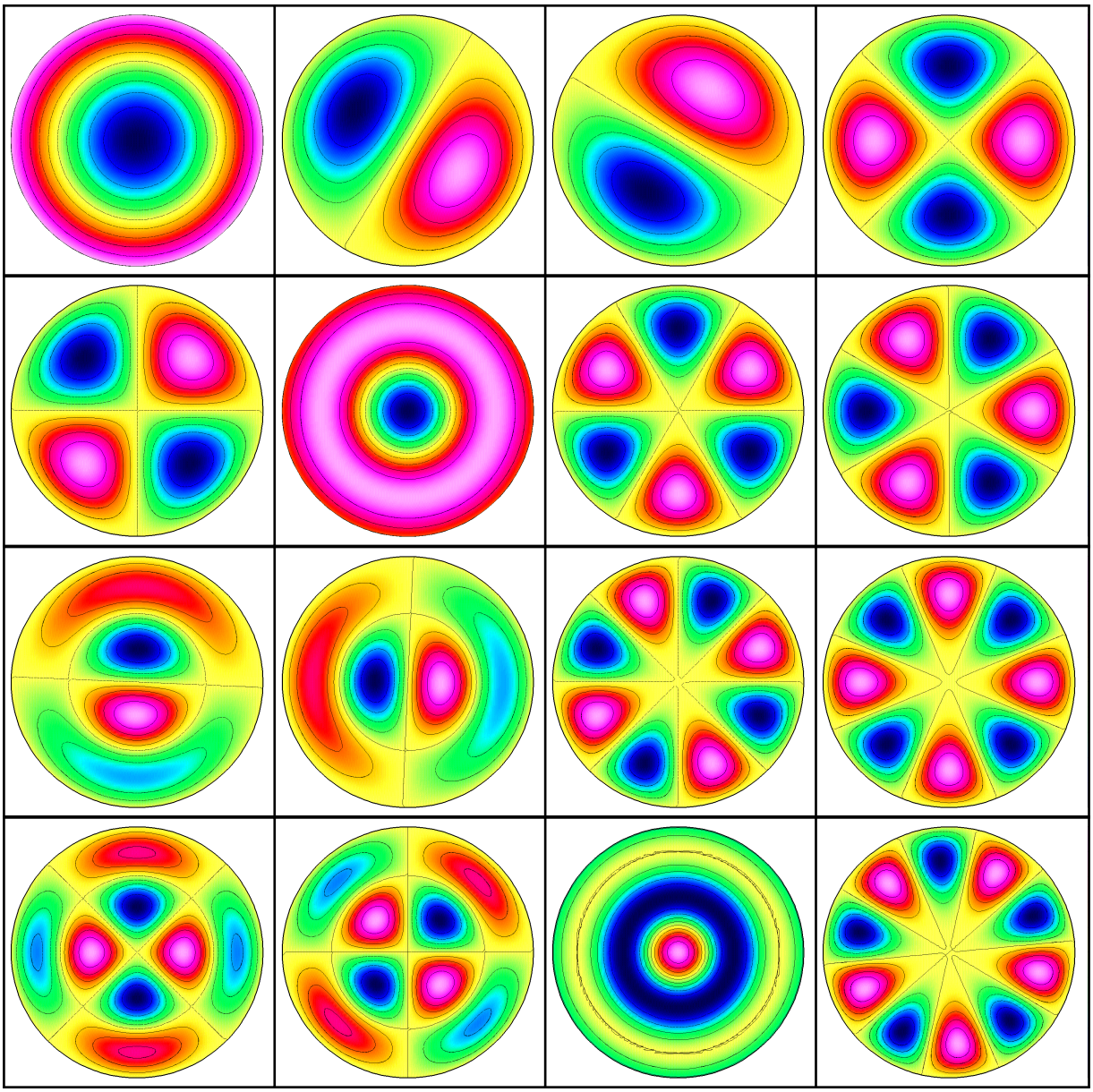

Figure 6: Computed eigenfunctions of a disk, arranged by magnitude of the eigenvalues.

4.4. Computed eigenfunctions of an ellipse in a box

Here are some computed eigenvalues for the ellipse in a box. There seem to be no multiple eigenvalues.

```
genEigs -noplot eigs.cmd -problem=laplace -eigCase=disk -g=ellipsee4.order4.s1.5.hdf -eigOption=1
  -eps_s=largest_magnitude -numEigenValues=68 -numEigenVectors=64 -tol=1.e-6 -bc1=d -show=eib64.show

Eigenvalue   0 : k=    3.3579349838 + (-0) I,
Eigenvalue   1 : k=    3.7172899474 + (-0) I,
Eigenvalue   2 : k=    4.2746640412 + (-0) I,
Eigenvalue   3 : k=    5.1551742507 + (-0) I,
Eigenvalue   4 : k=    6.7306843897 + (-0) I,
Eigenvalue   5 : k=    8.3202600219 + (-0) I,
Eigenvalue   6 : k=    8.8427800346 + (-0) I,
Eigenvalue   7 : k=   10.3307519679 + (-0) I,
Eigenvalue   8 : k=   12.3634449757 + (-0) I,
Eigenvalue   9 : k=   12.5520770161 + (-0) I,
Eigenvalue  10 : k=   12.9907804080 + (-0) I,
Eigenvalue  11 : k=   13.4171548012 + (-0) I,
Eigenvalue  12 : k=   14.6996193850 + (-0) I,
Eigenvalue  13 : k=   16.9199964707 + (-0) I,
Eigenvalue  14 : k=   17.3127037296 + (-0) I,
Eigenvalue  15 : k=   18.1024666484 + (-0) I,
Eigenvalue  16 : k=   18.5014568730 + (-0) I,
Eigenvalue  17 : k=   20.6358025973 + (-0) I,
Eigenvalue  18 : k=   21.0111189039 + (-0) I,
Eigenvalue  19 : k=   21.4286701882 + (-0) I,
Eigenvalue  20 : k=   23.2851814358 + (-0) I,
Eigenvalue  21 : k=   24.7564697513 + (-0) I,
Eigenvalue  22 : k=   24.9029841436 + (-0) I,
```

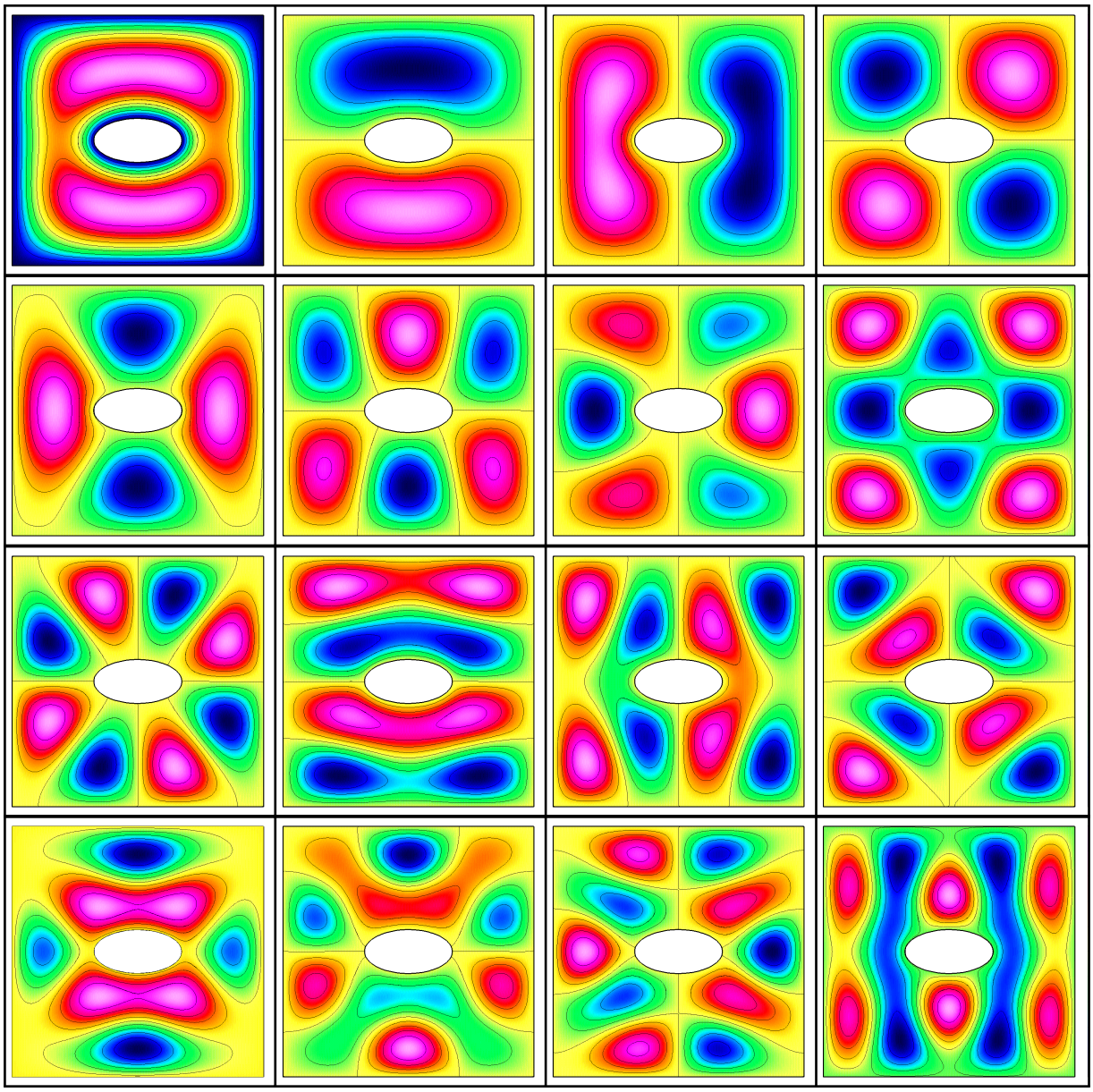

Figure 7: Computed eigenfunctions of an ellipse in a box, arranged by magnitude of the eigenvalues.

4.5. Computed eigenfunctions for some shapes in a box

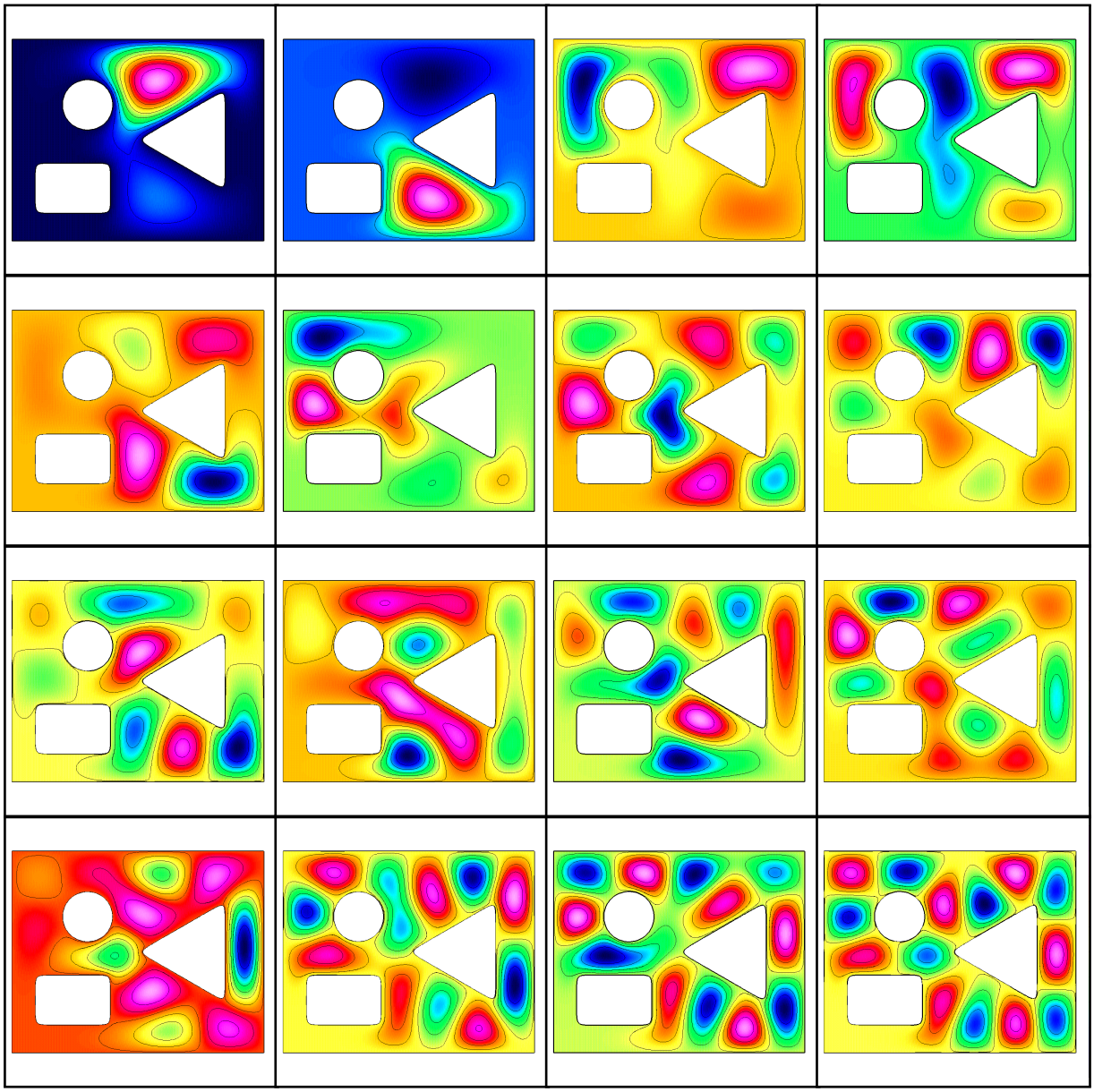

Figure 8: Computed eigenfunctions of some shapes in a box, arranged by magnitude of the eigenvalues.

4.6. Scattering from a cylinder

We consider scattering of a Gaussian plane wave from a cylinder (periodic in y). Figure 9 shows some eigenfunctions.

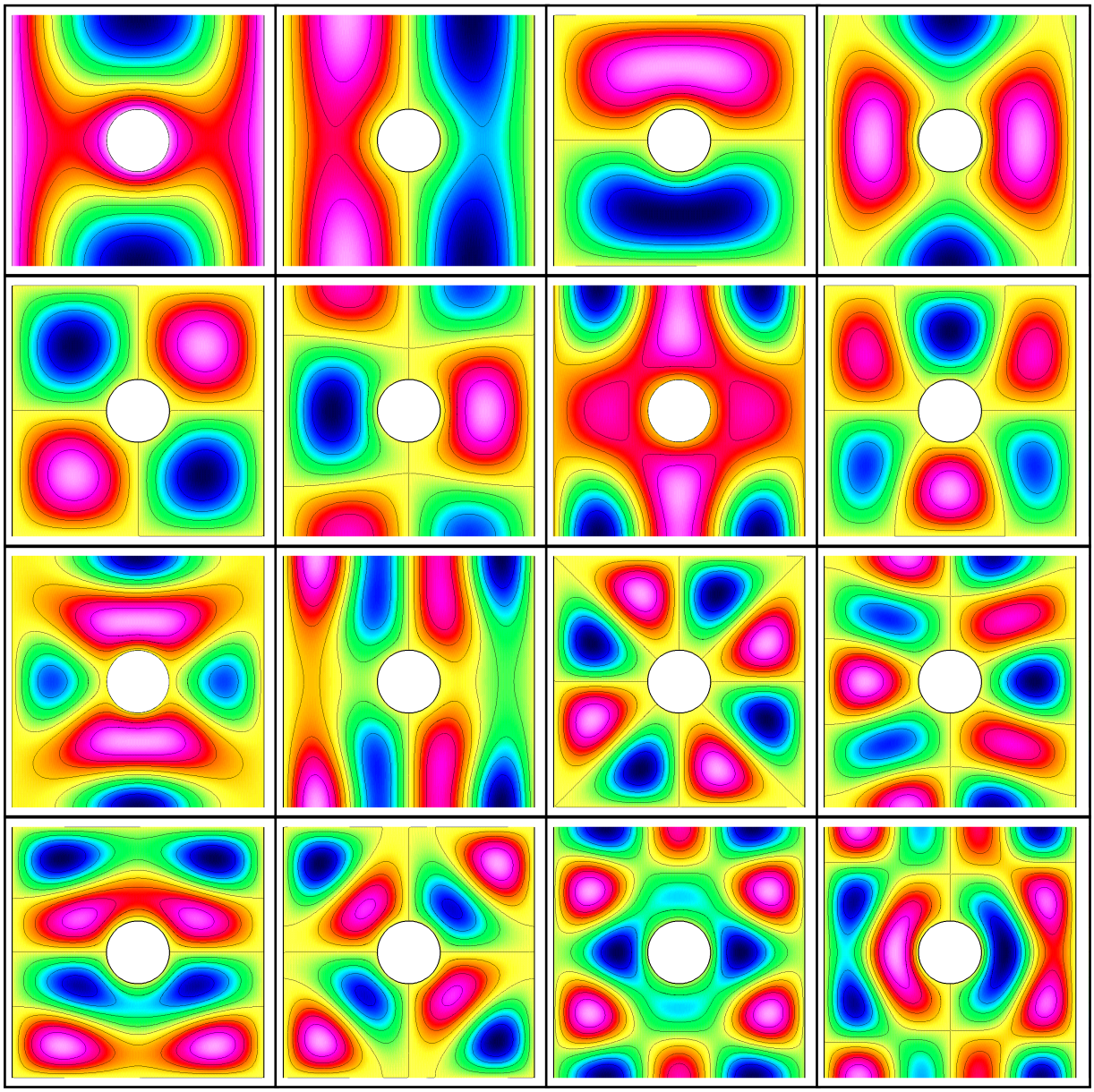

Figure 9: Computed eigenfunctions of a cylinder in a box, arranged by magnitude of the eigenvalues.

Here are commands to compute the scattering of a Gaussian plane wave from the cylinder.

```
eveSolver square.cmd -eigFile=cicEigs1024.show -beta=40 -x0=-1.25 -y0=0.5 -k0=0 -ic=gpw
Errors in u0 fit: max-err=1.28e-03, l2-err=3.78e-04
Errors in u1 fit: max-err=4.13e-02, l2-err=1.28e-02
```

2048 modes:

```
eveSolver square.cmd -eigFile=cicEigs2048.show -beta=40 -x0=-1.25 -y0=0.5 -k0=0 -ic=gpw
Errors in u0 fit: max-err=2.68e-04, l2-err=3.79e-05
Errors in u1 fit: max-err=3.97e-03, l2-err=7.25e-04
```

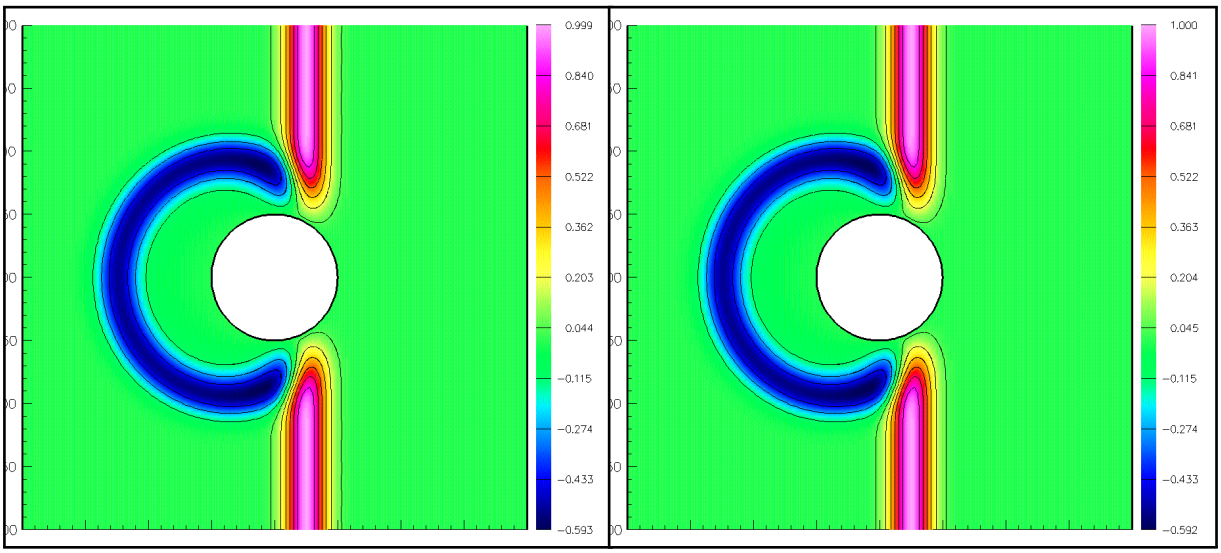

Figure 10: Scattering from a cylinder using 1024 eigenfunctions, $t = 1.5$. Left: 1024 modes, right 2048 modes.

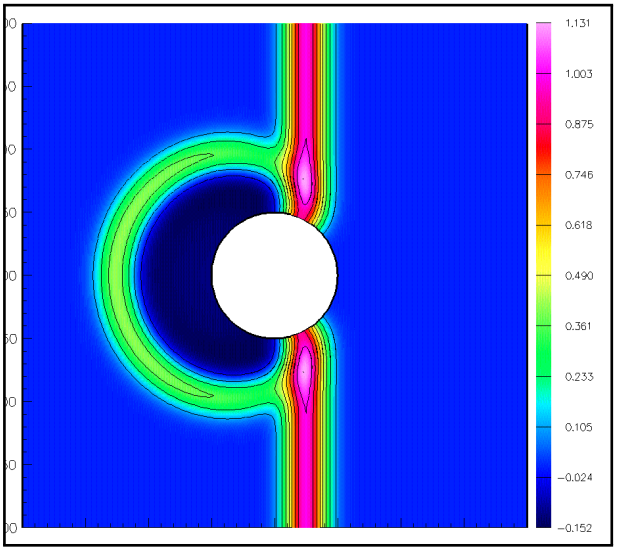

Figure 11: Neumann BC: scattering from a cylinder, $t = 1.5$. Left: 1024 modes, right 2048 modes.

4.7. RPI Scattering

We consider scattering of a Gaussian plane wave from three bodies with shapes of the letters R, P and I. Figure 14 shows some eigenfunctions.

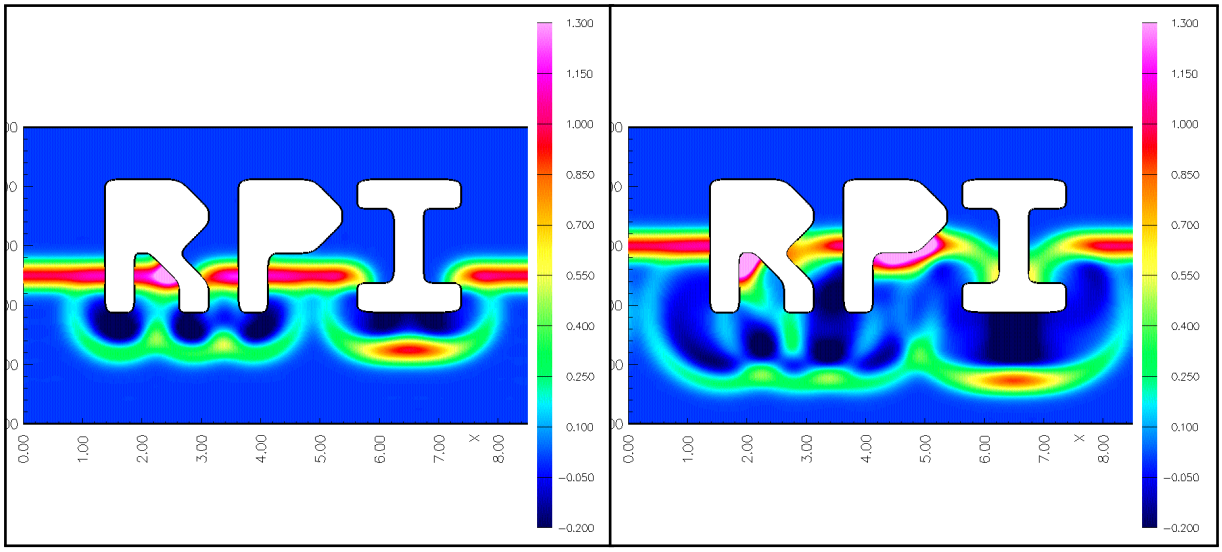

Figure 12: Scattering from the letters RPI, $t = 1.5$, 1024 modes. Neumann BCs on letters. Gaussian plane wave travels upward from the bottom.

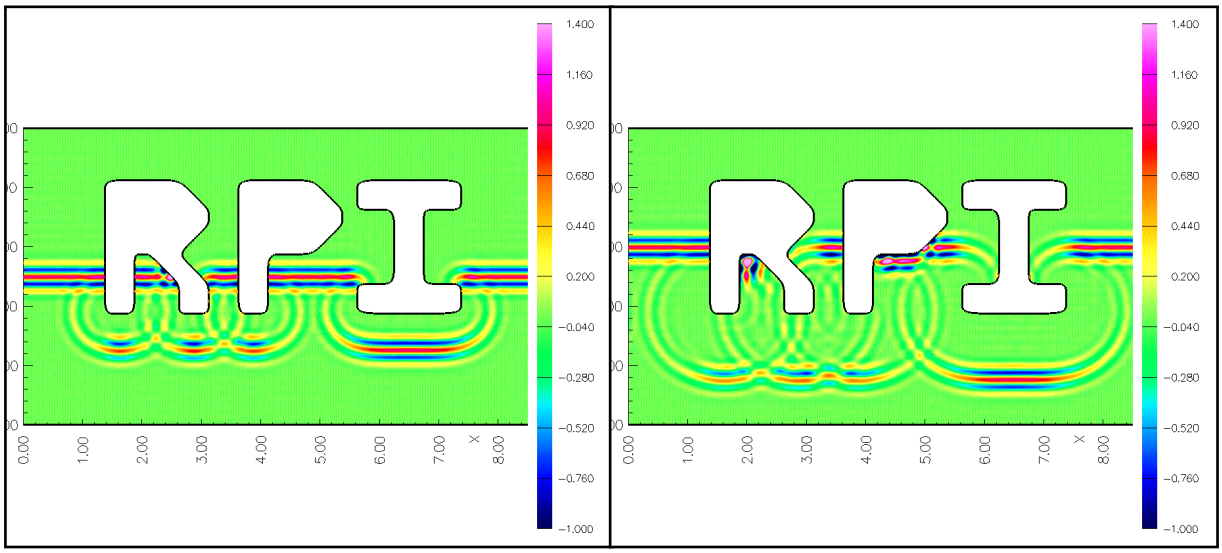

Figure 13: Scattering from the letters RPI, $t = 1.5$, 4096 modes. Neumann BCs on letters. Modulated Gaussian plane wave travels upward from the bottom.

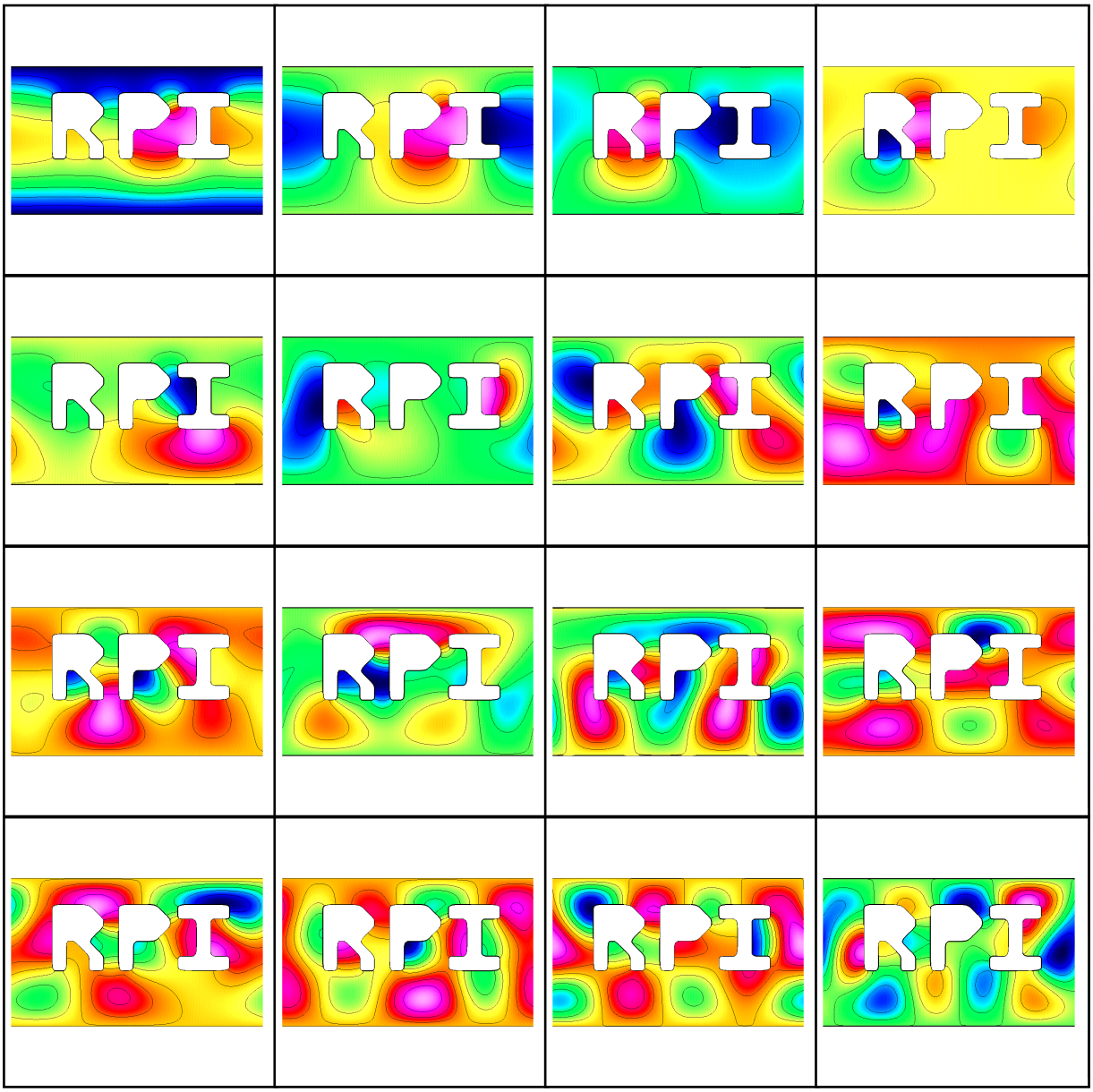

Figure 14: Computed eigenfunctions for the letters RPI, arranged by magnitude of the eigenvalues. Outer boundary is periodic in x and Dirichlet on top/bottom, Neumann BCs on the letters.

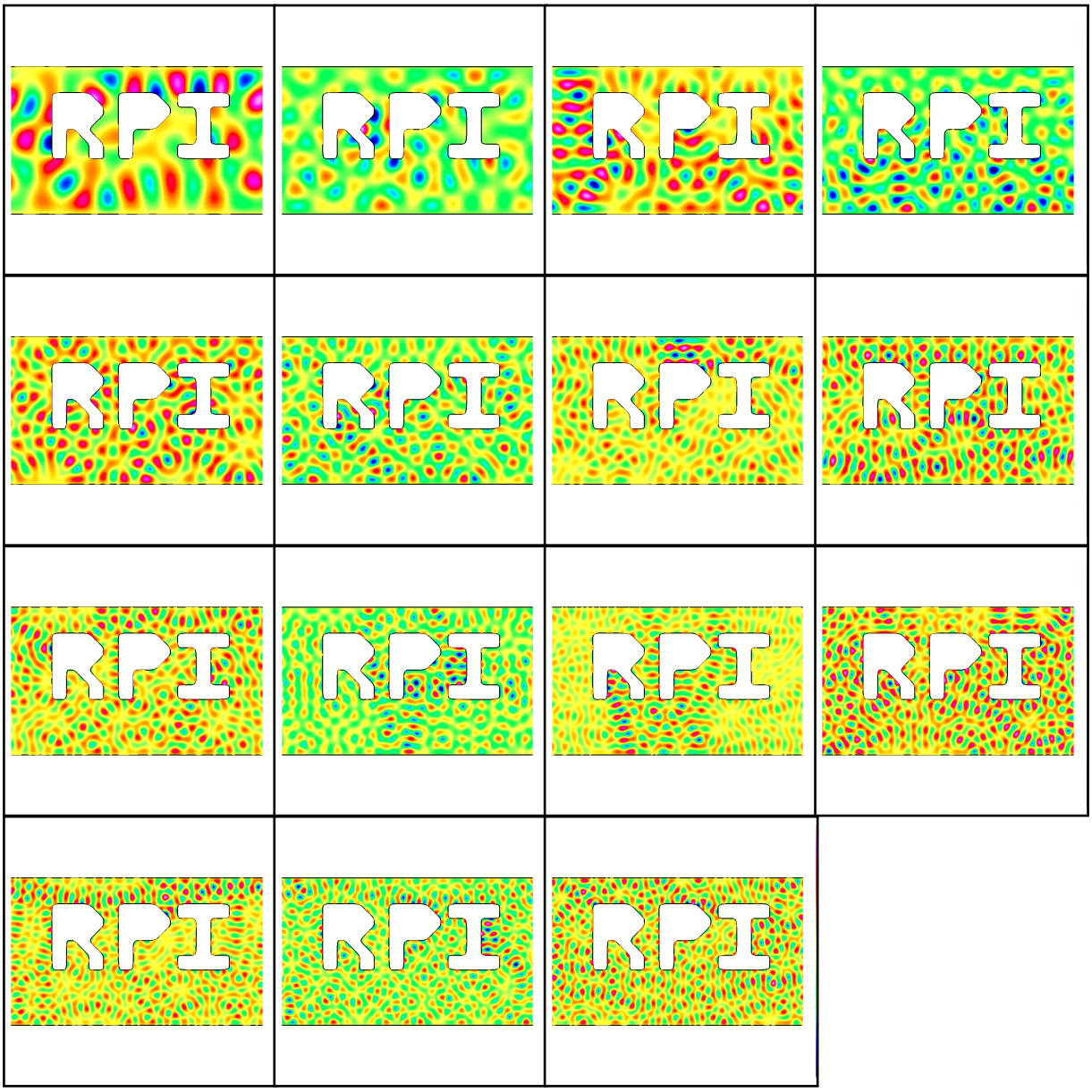

Figure 15: Selected computed eigenfunctions for the letters RPI, arranged by magnitude of the eigenvalues. Outer boundary is periodic in x and Dirichlet on top/bottom, Neumann BCs on the letters. Modes 64, 128, ...