`MPSpack` tutorial

Alex Barnett*†

April 13, 2016

Abstract

This tutorial shows how a variety of two-dimensional Laplace and Helmholtz boundary-value and eigenvalue problems may be numerically solved simply and accurately with the `MPSpack` toolbox for MATLAB. We assume basic familiarity with MATLAB and with linear partial differential equations.

Contents

*Department of Mathematics, Dartmouth College, Hanover, NH, 03755, USA; and Simons Center for Data Analysis, Simons Foundation, NY, NY, 10011

†Timo Betcke co-conceived, and co-authored in the period 2008–2009.

1 About this tutorial

This tutorial is designed for 'bottom-up' learning of the features of `MPSpack`, i.e. by progressing through simple examples. In that sense it complements the user manual which describes the theoretical framework in broad strokes and therefore could be considered 'top-down'. We will skip most of the mathematics behind the techniques, focusing on computing and plotting useful PDE solutions. We hope you will try each command as you read!

Throughout we will identify the plane $\mathbb{R}^2$ with the complex plane $\mathbb{C}$, by the usual map $z = x+iy$. In other words $(2,3)$ and $2+3i$ represent the same point. We use `teletype` font to designate commands that may be typed at the MATLAB prompt. You may get help on any `MPSpack` command by typing `help command`. All the code examples in this document, including code to generate the figures, is found in the `examples/` directory. The codes for Sections 2–8 are named `tut_*.m` according to the section names in `tutorial.tex`.

Computation times are reported for a laptop (either Intel Core Duo from 2006, or i7-3720QM from 2012).

I have yet to add a tutorial explanation of the Barnett–Hassell–Tacy methods for Neumann EVPs; to appear shortly.

2 Solving Laplace's equation in a disc

We start by setting up a domain in $\mathbb{R}^2$. Domains are built from segments which define their boundary. To make the unit disc domain, we first need a circle segment with center 0, radius 1, and angle range $[0, 2\pi)$, as follows,

```
s = segment([], [0 1 0 2*pi])
```

The object `s` is indeed a circular segment, as we may check by typing `s.plot`, producing Fig. 1a. All segments have a natural *sense*, i.e. direction of travel:

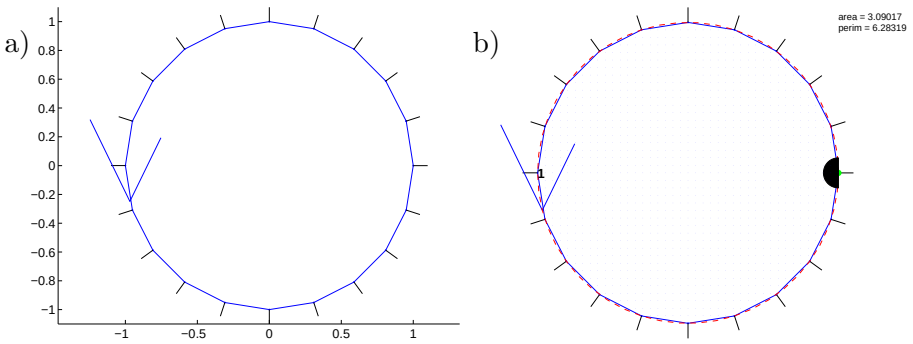

Figure 1: a) circular closed segment, b) unit disc domain. Both have a periodic trapezoidal quadrature rule with $M = 20$ quadrature points

for this segment it is counter-clockwise, as shown by the downwards-pointing arrow symbol overlayed onto the segment at about 9 o'clock.[1] Notice also normal vectors (short 'hairs') pointing outwards at each boundary point; our definition is that normals on a segment always point to the *right* when traversing the sense of the segment.

We create the domain interior to this segment with

```
d = domain(s, +1)
```

where the second argument (here +1, the only other option being −1) specifies that the domain is to the 'standard' side of the segment, which we take to be such that the normals point *away from* the domain. That is, with +1 the domain lies to the *left* of the segment when traversed in its correct sense (with −1 the domain would lie to the right of the segment.) Typing `d.plot` produces[2] Fig. 1b. Note that perimeter and area are automatically labelled (these are only rough approximations intended for sanity checks).

Laplace's equation $\Delta u = 0$ is Helmholtz's equation with wavenumber zero, which we set for this domain with,

```
d.k = 0;
```

If the problem has many domains the wavenumber should be set globally using the method `problem.setoverallwavenumber`. Our philosophy is to approximate the solution in the domain by a linear combination of *basis functions*, each defined over the whole domain. We choose 8th-order har-

[1] In fact, segments are parametrized internally as function $z(t)$ of a real variable $t \in [0, 1]$, and the sense is the direction of increasing t. Segment `s` stores this function as `s.Z`.

[2] There are extra plotting options and features that are described in documentation such as `help domain.plot`. E.g. in this figure a grid of points interior to the domain has been included, achieved with `opts.gridinside=0.05; d.plot(opts);`

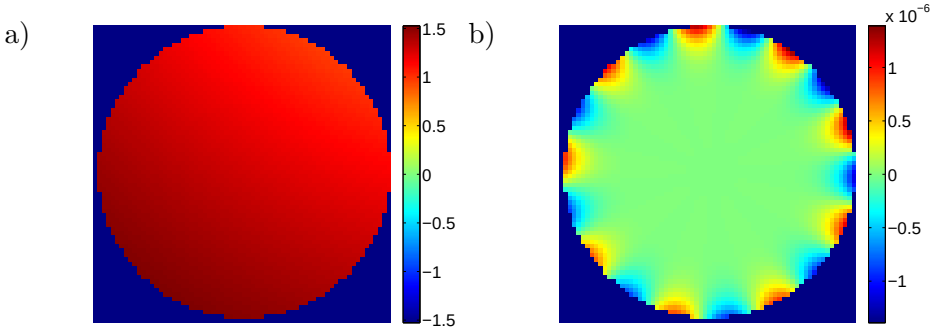

Figure 2: a) Numerical solution field u, b) pointwise error $u-f$, for Laplace's equation in the unit disc with $M = 20$ quadrature points and 8th-order harmonic polynomials.

monic polynomials $u(z) = \sum_{n=0}^{8} c_n \operatorname{Re} z^n + \sum_{n=1}^{8} c_{-n} \operatorname{Im} z^n$, where $\boldsymbol{c} := \{c_n\}_{n=-8}^{8} \in \mathbb{R}^{17}$ is a coefficient vector, based at the origin 0, using the command

```
d.addregfbbasis(0, 8);
```

Let's specify Dirichlet boundary data $f(z) = \ln|z-2-3i|$ for z on the segment[3] by representing this as an anonymous function `f` and associating it with one side of the segment,

```
f = @(z) log(abs(z-2-3i));
s.setbc(-1, 'd', [], @(t) f(s.Z(t)));
```

Note that we needed to pass in a function not of location z, but of the segment parameter t; this was achieved by wrapping `f` around the parametrization function `s.Z`. The first argument -1 expresses that the boundary condition is to be understood in the limit approaching from the side *opposite* the segment's normal direction, which is where the domain is located. The second argument `'d'` specifies that the data is Dirichlet.

Finally we use the domain to make a boundary-value problem object `p`,

```
p = bvp(d);
```

and may then solve (in the least-squares sense) a linear system for the coefficients

```
p.solvecoeffs;
```

If it is needed, `p.co` now contains the coefficients vector $\boldsymbol{c}$. To evaluate and plot the solution we simply use,

[3] In other words, $f(x,y) = \ln\sqrt{(x-2)^2+(y-3)^2}$ for points (x,y) on the boundary.

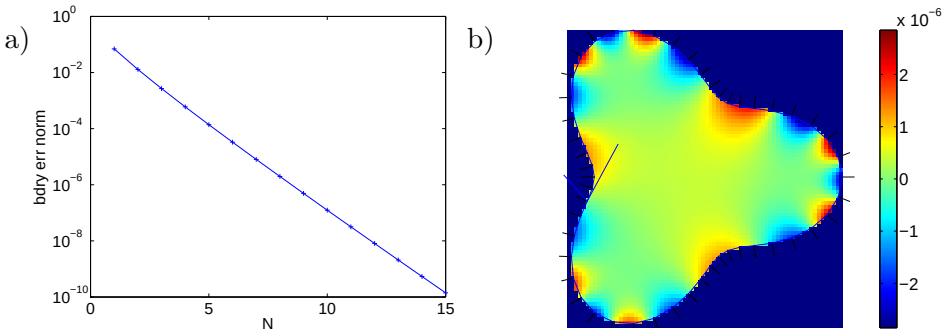

Figure 3: a) Convergence of boundary error L^2 norm for harmonic polynomials for Laplace equation in the unit disc, b) solution error for same boundary data f in a smooth star-shaped 'trefoil' domain (normals also shown).

```
p.showsolution;
```

The software chose an appropriate grid covering the domain (points outside the domain are made transparent, but show up deep blue in this document), giving Fig. 2a.

3 Accuracy, convergence, and smooth domains

How accurate was our numerical solution u? One measure is the L^2 error on the boundary, and is estimated by

```
p.bcresidualnorm
```

which returns 2.09×10^{-6}. However, since the function $f(z)$ is already harmonic in the domain, it is in fact the unique solution, and we may plot the pointwise error in u by passing in the analytic solution as an option,

```
opts.comparefunc = f; p.showsolution(opts);
```

giving Fig. 2b. Note that the color scale is 10^{-8}.

In the above, boundary integrals were approximated using the default of $M = 20$ quadrature points, barely adequate given the oscillatory error function in Fig. 2b. M may be easily changed either by specifying a non-empty first argument in the `segment` constructor above, or for an existing segment as follows,

```
s.requadrature(50); p.solvecoeffs; p.bcresidualnorm
```

which now gives 1.98×10^{-6}, not much different than before. Notice that we did not have to redefine the domain `d` nor the BVP object `p`.

Exploring the convergence of the boundary error norm with the basis set

order needs a simple loop over N,

```
for N=1:15
  p.updateN(N); p.solvecoeffs; r(N) = p.bcresidualnorm;
end
figure; semilogy(r, '+-'); xlabel('N'); ylabel('bdry err norm');
```

As the resulting Fig. 3a shows, the convergence is exponential.[4] Notice we used `updateN` to change the basis set degree in a problem (in this simple case it is equivalent to `d.bas{1}.N = N;`)

Say we want to change the shape of segment `s`, to a smooth star-shaped 'trefoil' domain expressed as by radius $R(\theta) = 1 + 0.3\cos 3\theta$ as a function of angle $0 \le \theta < 2\pi$. This is achieved by passing a 1-by-2 cell array containing the function R and its derivative $R' = dR/d\theta$ to a variant of the segment constructor,

```
s = segment.radialfunc(50, {@(q) 1 + 0.3*cos(3*q), @(q) -0.9*sin(3*q)});
```

We again chose $M = 50$. The analytic formula for R' is needed to compute normal derivatives to high accuracy.

One might ask: has this change to `s` *propagated* to the existing domain object `d` and BVP object `p`, which both refer to it? In contrast to the case of quadrature point number M above, the answer is no: `s` is overwritten by a newly-constructed object, while `d` and `p` still contain handles pointing to the *old* `s`. Furthermore, the fact that the segment had domain `d` attached to its 'minus' or back side has been forgotten, as have the boundary conditions. (These segment properties are described in the `MPSpack` user manual.) We must therefore rerun the code from Sec. 2 to construct `d` and `p` afresh, before solving.[5] The result, plotting the pointwise error as before, is shown by Fig. 3b for $N = 8$ and $M = 50$.

The `radialfunc` constructor above is limited to radial functions with quadrature equidistant in angle. Instead you may create a segment from arbitrary smooth parametrizations $z(t)$ for $t \in [0, 1]$, as long as $z'(t)$ is also given. For instance, a closed crescent-shaped analytic segment is produced (try it!) by,

[4] Asymptotically, error $\sim e^{-\alpha N}$. In fact the rate is $\alpha = \ln\sqrt{13}$, related to the conformal distance to the nearest singularity [4], which here is at $2 + 3i$.

[5] Note that in theory it would be possible to change one by one each of the segment properties, `t`, `w`, `speed`, etc, to define the new segment without changing its identity, but this is cumbersome. Similarly, searching and changing all references to a segment in the properties of `d` and `p` is cumbersome. Neither has been implemented by us since problem setup time is very rapid.

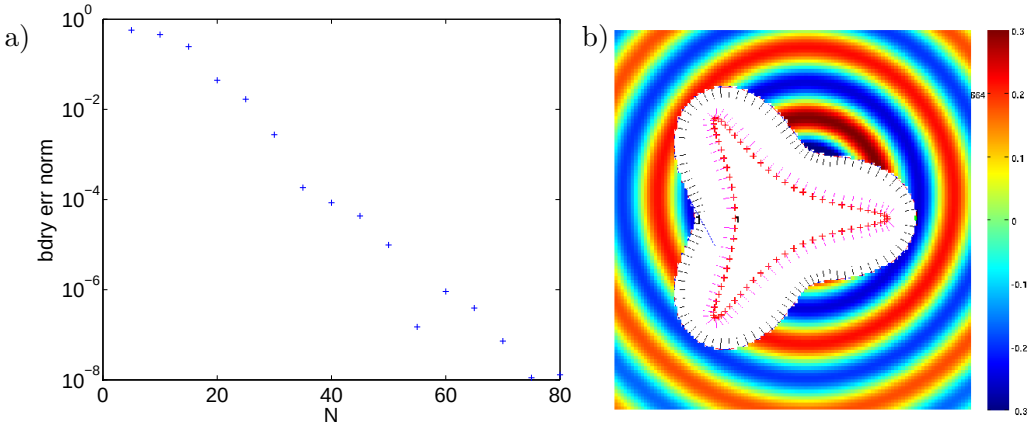

Figure 4: a) Convergence of exterior Helmholtz BVP with MFS basis, b) Domain boundary and MFS charge curve geometry, and (real part of) the solution field outside the domain.

```
a = 0.2; b = 0.8; w = @(t) exp(2i*pi*t);
s = segment(100, {@(t) w(t)-a./(w(t)+b), ...
                  @(t) 2i*pi*w(t).*(1 + a./(w(t) + b).^2)}, 'p');
```

Note that a convenient variable $w = e^{2\pi i t}$ was used via nested anonymous functions. Heed also the new final argument `'p'` which enforces periodic quadrature (the constructor doesn't try to guess your preferred rule). In order to get high-order (or spectral) convergence, it is recommended that you choose only smooth (or analytic) z. If periodic quadrature is used, this also applies to the 1-periodic extension of z to the real line. If $z(1) \neq z(0)$, the ends of the segment will not connect up, and the domain constructor above will report an error.

4 Helmholtz equation, exterior and multiply connected domains

Changing from the Laplace to Helmholtz equation is as simple as setting `d.k` to a positive value. We start a fresh example: an exterior Helmholtz BVP with Neumann boundary data, and the Sommerfeld radiation condition [6]. This has a unique solution.

The simplest unbounded domain is $\mathbb{R}^2$, which is created with

```
d = domain([], []);
```

One may check that its area `d.area` is ∞. Exterior domains can be created by excluding from $\mathbb{R}^2$ the interior of a closed segment, for instance the trefoil segment introduced above,

```
tref = segment.radialfunc(100, {@(q) 1 + 0.3*cos(3*q), @(q) -0.9*sin(3*q)});
d = domain([], [], tref, -1);   % overwrites previous d
```

Note the choice -1 for the direction argument, which states that the domain lies on the 'nonstandard' side of the segment, i.e. to the right side as the segment is traversed in its natural sense, with the segment normals pointing *into* the exterior domain. As before, we set up Dirichlet boundary data corresponding to a known radiative solution (a point source lying in the segment interior),

```
d.k = 10; f = @(z) besselh(0,d.k * abs(z-0.3-0.2i));
tref.setbc(1, 'D', [], @(t) f(tref.Z(t)));
```

A convenient basis set for radiative solutions is a set of fundamental solutions ('MFS basis') with origins $\mathbf{y}_j$ lying on a closed curve interior to the segment. The formula for the jth basis function at location $\mathbf{x} \in \mathbb{R}^2 \setminus \mathbf{y}_j$ is $\Phi(|\mathbf{x}-\mathbf{y}_j|)$, where the fundamental solution for the Helmholtz equation at wavenumber k is $\Phi(\mathbf{x}) = \frac{i}{4}H_0^{(1)}(k|\mathbf{x}|)$. We set this up and plot convergence of boundary error norm,

```
opts.tau = 0.06; d.addmfsbasis(tref, [], opts);
p = bvp(d);
for N=5:5:80,
  p.updateN(N); p.solvecoeffs; r(N) = p.bcresidualnorm;
end
figure; semilogy(r, '+-'); xlabel('N'); ylabel('bdry err norm');
```

This gives the convergence plot Fig. 4a, and executing `d.plot; p.showbasesgeom; p.showsolution;` gives Fig. 4b. Notice that the MFS charges lie some distance interior to the curve—this is controlled by the `opts.tau` parameter which makes use of the fact that the segment is specified as an analytic function and hence generates a new curve when the parameter t is displaced by τ in the imaginary direction.[6] Plotting pointwise error with

```
opts.comparefunc = f; figure; p.showsolution(opts);
```

shows that is it around 10^{-13}.

[6] A general rule is that for good spectral convergence the MFS curve should be as far as possible from the solution domain boundary, while still 'shielding' this boundary from singularities in the analytic continuation of the solution field. The user will have to adjust this parameter, although it should not depend on k. See [1].

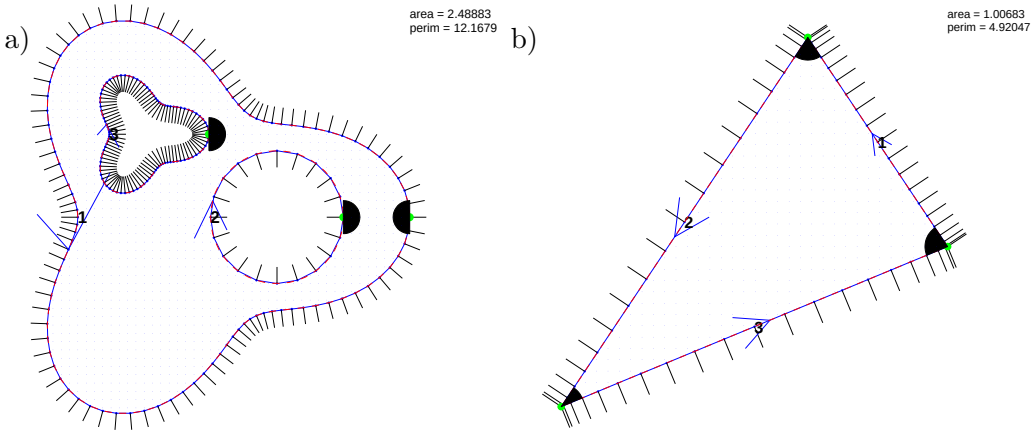

Figure 5: a) A multiply-connected domain. b) A polygonal domain.

A non-simply connected domain may be built by specifying excluded regions from a simply connected bounded domain. For example, to remove from an interior trefoil a circular 'hole',

```
tref.disconnect;  % clears any domains from segment
c = segment([], [0.5 0.4 0 2*pi]); % new circular segment
d = domain(tref, 1, c, -1);
```

Note that segment `tref` had previously been 'linked' to the old domain `d` at the start of this section, hence the need to 'disconnect' it (or create a fresh segment) before building new domains from it. If the direction signs $+1$ and -1 are not correct as above, an error is reported (check this!) We may exclude two regions as follows, where the new region is a smaller copy of the trefoil,

```
tref.disconnect; c.disconnect;
smtref = tref.scale(0.3);    % create new rescaled tref copy
smtref.translate(-0.3+0.4i); % move the segment smtref
d = domain(tref, 1, {c smtref}, {-1 -1});
```

Typing `d.plot` gives Fig. 5a. Notice that the domain's boundary is the union of three segments. They are labeled 1, 2, and 3, showing the order in which segment handles are stored internally in the domain object `d.seg`. The convention for plotting domains is that the normals are those of the domain, rather than the normal intrinsic to each segment. The figure shows all normals pointing away from the domain, as it should. Similarly, the

arrow directions are modified by the signs $(1, -1, -1)$ that were passed in, so that, following the arrows the domain always lies to the *left*. (The black semicircles will be explained in the next section.)

More complicated domains similar to the above will be demonstrated later in Secs. 8 and 10.

5 Polygons, corners, and corner-adapted bases

So far each disconnected boundary piece of the domain has been a single segment connected to itself head-to-tail. More generally, a *list* of segments may be used to create these closed boundary pieces, as long as the last in the list connects back to the starting point of the first. For instance, a triangle is built by sending a list of three line segments to the domain command; such a list can simply be built from a list of vertices as follows,

```
s = segment.polyseglist([], [1 exp(3i*pi/8) exp(5i*pi/4)]);
tri = domain(s, 1);
```

Since `s` is a 1-by-3 array of (handles to) segments, the 2nd argument of the domain constructor is automatically vectorized to `[1 1 1]`. Each of the segments could have been made separately, e.g. the first by `s(1) = segment([], [1 exp(3i*pi/8)]);` etc. Fig. 5b shows `tri.plot` output for this domain.[7]

Let's discuss corners: their angles are indicated by the solid black 'fans' in the plot (before now these have had angle π, hence semicircles). A black fan at the junction of two segments indicates that a corner linkage was made (warnings will be given if any segment ends are left dangling), and shows the angle range pointing *into* the domain.

Segment lists may also be sent to the excluded-region arguments of the domain constructor, for instance to create the domain exterior to the triangle,

```
exttri = domain([], [], s(end:-1:1), -1);
```

where it was important to reverse the order of the segment list so that they connect head-to-tail correctly. To create the domain exterior to two nearby triangles,

[7] Notice that periodic quadrature would now be inappropriate: in fact $M = 20$ point Clenshaw-Curtis is used by default for open segments

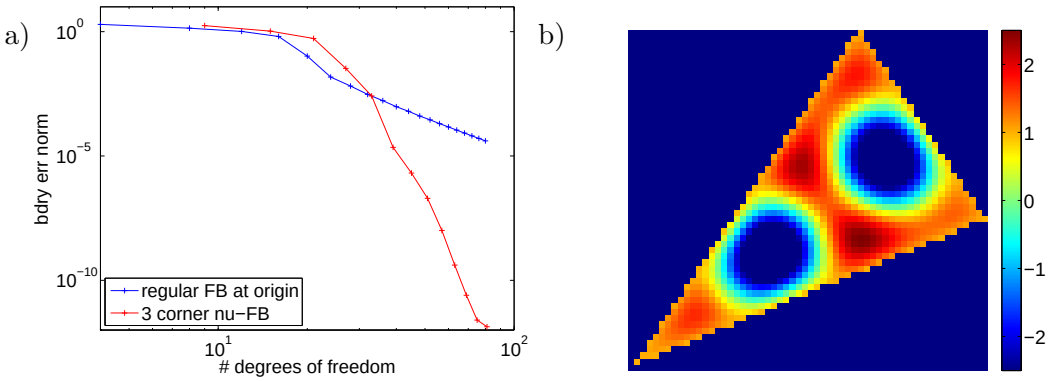

Figure 6: a) Comparing convergence of boundary error norm for Fourier Bessel vs corner-adapted fractional-order Fourier Bessel basis sets in a triangle with unity Dirichlet boundary data, for Helmholtz BVP with $k = 10$, on log-log axes. b) The solution function.

```
ss = s.translate(2);
exttwotri = domain([], [], {s(end:-1:1), ss(end:-1:1)}, {-1, -1});
```

We remind the reader that to create the above domains using the existing segment array `s`, each time a `s.disconnect` would be needed beforehand (this acts on all segments in the list).[8] A universal rule is:

Each side of a segment may be associated with at most one domain

Say we want to solve an interior Helmholtz BVP on the original triangle of this section, and `s` and `tri` have been set up as with the first two commands of this section but with $M = 50$ quadrature points. Say we want constant boundary data 1. We may use a Fourier-Bessel basis set, solve, and plot error convergence with a simple code,

```
s.setbc(-1, 'D', [], @(t) ones(size(t)));   % note inline "1 function"
tri.addregfbbasis(0, []); tri.bas{1}.rescale_rad = 1.0; % for stability
p = bvp(tri);
tri.k = 10;                                 % set wavenumber
Ns = 2:2:40; for i=1:numel(Ns)
  p.updateN(Ns(i)); p.solvecoeffs; r(i) = p.bcresidualnorm;
```

[8] Also note that the domains previously linked to the segments, such as `tri`, would be left 'dangling' since `s` no longer is linked back to them. Attempts to use `tri` in a BVP would now be doomed unless the data `s(:).dom` were manually rewritten to point to the domain (see manual). When in doubt, disconnect segments then remake all domains.

```
end
figure; loglog(2*Ns, r, '+-'); xlabel('# dofs'); ylabel('bdry err norm');
```

This produces the algebraic convergence shown in blue in Fig. 6a. Why is this so slow? The triangle has three *singular* corners (i.e. one whose angle is not π divided by an integer), and generically the exact solution to such a problem has a singularity at each such corner.[9]

We now show how corner-adapted basis sets achieve superior convergence and hence more efficiency. We clear the previous basis set from the domain then add fractional-order (i.e. 'wedge' expansion) Fourier-Bessel bases at each corner, of both cos and sin type.[10] Repeating the convergence study and comparing against the previous data is easy,

```
tri.clearbases;
opts.rescale_rad = 2.0; tri.addcornerbases([], opts);
r = []; Ns = 1:13; for i=1:numel(Ns)
  p.updateN(Ns(i)); nn(i) = p.N; p.solvecoeffs; r(i) = p.bcresidualnorm;
end
hold on; loglog(nn, r, 'r+-');  % plot error vs total # dofs
```

The new convergence is shown in red in Fig. 6a, and is much faster; in fact it appears superalgebraic [4]. The solution field is Fig. 6b, and its large values shows that we are quite close to a Dirichlet resonance of the domain. The basis set geometry in the domain can be visualized by `tri.showbasesgeom` which shows wedges implied by the corner expansions.

6 Scattering and transmission problems

Scattering of scalar waves (acoustic, or suitably-polarized Maxwell equations) involves finding a Helmholtz solution $u = u_{inc} + u_s$ with homogeneous boundary conditions on an obstacle, given an incident (often plane-wave) Helmholtz solution u_{inc}. The unknown u_s is then the solution to the radiative exterior BVP solved in Sec. 4 (see Sec. 10.3.1 for more detail), with inhomogenous boundary data related to u_{inc}.

We designed a `scattering` class to implement this in a user-friendly fashion. Creating the analytic `tref` segment (with $M = 250$) and its exterior domain `d` as in the start of Sec. 4, we set up and solve the sound-soft scattering problem as follows,

[9] The power law for convergence is related to the corner angles and is discussed in [7].

[10] Sometimes an expansion is needed at some corners and not others. E.g. expansions at only the first two corners can be set up by passing in the options `opts.cornermultipliers = [1 1 0];`.

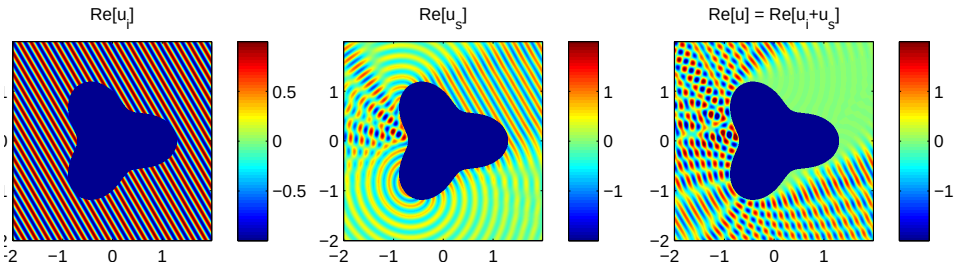

Figure 7: Sound-soft (homogeneous Dirichlet BC) scattering from a smooth trefoil at $k = 30$ using an MFS basis. Left: incident wave u_{inc}. Center: scattered wave u_s. Right: their sum, the total solution field u.

```
opts.tau = 0.05; d.addmfsbasis(tref, 200, opts);  % basis set for ext domain
tref.setbc(1, 'D', []);                 % homogeneous Dirichlet BCs: sound-soft
p = scattering(d, []);
k = 30; p.setoverallwavenumber(k);
p.setincidentwave(pi/6);                % incident plane wave with given angle
p.solvecoeffs;                          % fills matrices, least-squares soln
```

The quadrature approximation to the L^2 boundary error is 3×10^{-10} as given by `p.bcresidualnorm`; exponential convergence could be checked with a little loop as in Sec. 3. The incident wave, scattered wave, and solution are plotted[11] as in Fig. 7 by typing `p.showthreefields`. The solution took around 0.1 sec; the time for plotting is around 1 sec, but depends on the grid resolution requested. A good choice of the `opts.tau` distance parameter must usually found by trial and error, examining the charge point locations and solution quality.

We may trivially switch to other physical boundary conditions by replacing the `setbc` command with either of the following,

```
tref.setbc(1, 'N', []);                 % homogeneous Neumann: sound-hard
tref.setbc(1, 1i*k, 1);                 % homogeneous Robin: impedance
```

In the latter case a boundary condition $iku + u_n = 0$ is applied, which strongly absorbs incident waves.

A transmission problem involves coupled regions of different wavenumber, for instance wavenumber k in an exterior domain, and wavenumber nk

[11] We in fact tweaked the grid spacing and bounding-box options via `o.dx=0.01; o.bb = 2*[-1 1 -1 1]; p.showthreefields(o);`

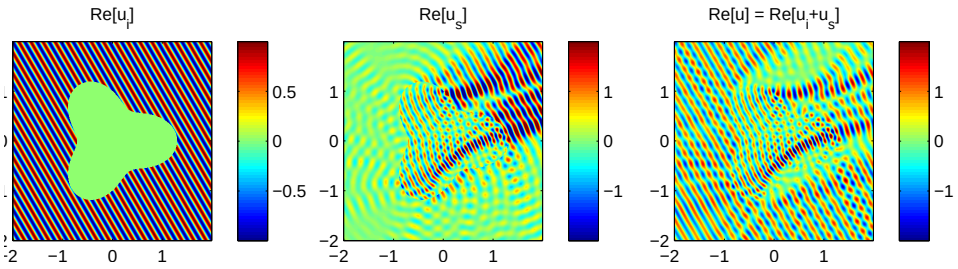

Figure 8: Transmission scattering from a smooth trefoil with refractive index (interior to exterior wavenumber ratio) 1.5 and exterior wavenumber $k = 30$, using MFS bases inside and out. Left: incident wave u_{inc}. Center: scattered wave u_s. Right: total solution field u.

in its complement, where n is the 'refractive index'[12] of the interior. Reusing the above domain `d` and its basis, we need a new interior domain `di` and its MFS basis (with charge points this time outside the boundary curve),

```
di = domain(tref, 1); di.refr_ind = 1.5;
opts.tau = -0.03; di.addmfsbasis(tref, 220, opts);
tref.setmatch('diel', 'TM');
p = scattering(d, di);                % d is an 'air' domain, di not air domain
p.setoverallwavenumber(k);
```

The matching condition that u and u_n are continuous across the interface is appropriate for TM-polarized Maxwell (in two dimensions, i.e. z-invariant), for a dielectric problem, hence the `setmatch` arguments. Then, `p.solvecoeffs; p.showthreefields;` produces Fig. 8, which has error in boundary matching (summing the L^2 errors in the jump in u and jump in u_n) of 10^{-6} given by `p.bcresidualnorm`. It can be verified that the convergence is exponential; MFS bases are an excellent choice for analytic boundaries.[13]

Notice that the `scattering` constructor takes *two* arguments (recall `bvp` took only one, the domain or list of domains in the problem). The first argument is the (list of) 'air' (as opposed to dielectric) domains, the second the 'non-air' domains. The former are interpreted as the domains which have the incident plane wave as their u_{inc}; notice the non-air domain `di` has trivial u_{inc} in the left plot of Fig. 8, because this index $n \neq 1$ domain does not have the incident plane wave as a Helmholtz solution of the correct

[12] Here we borrow notation from the optical application.

[13] If condition number is not a problem [1]. For well-conditioned formulations, layer potentials are needed as described in the next section.

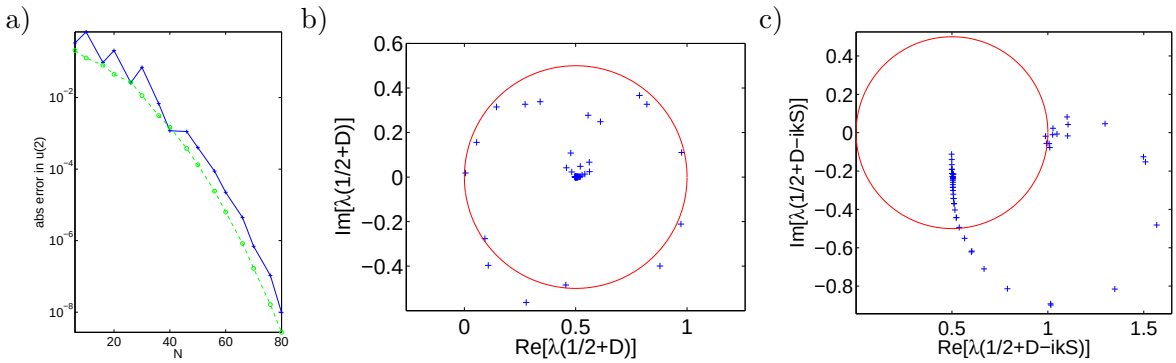

Figure 9: a) Pointwise convergence of exterior Helmholtz Dirichlet BVP with double-layer representation, at $k = 10$ (solid: pure DLP, dotted, combined-field formulation). b) Spectrum of $(\frac{1}{2}I + D)$ in the complex plane (for $M = 80$). We have superimposed a circle of radius $1/2$ centered at $1/2$, to which the eigenvalues appear to condense in the $k \to \infty$ limit—to prove this is an open problem! c) Spectrum of $(\frac{1}{2}I + D - ikS)$ relevant for combined-field formulation.

wavenumber nk. In the above, there was no freedom of choice of the air/non-air categorization, but in more complicated problems the exterior region may be divided into subdomains, some of which are passed in as air domains, others not; see Sec. 10.3. This enables dielectric-coated metals, dielectrics with interior holes, etc, to be solved; see Sec. 10.2.

7 Layer potentials

Layer potentials are representations of Helmholtz solutions involving a weighted integral of a fundamental solution over a boundary [6]. They are similar to the MFS representations discussed above, with the crucial advantage that a *second-kind* formulation is often possible, i.e. the operator involved is identity plus compact, and the resulting matrix problems are well-conditioned. It is easy to set up single- and double-layer potentials (SLP and DLP) living on a segment. Starting with the trefoil segment `tref` defined in Sec. 4, a DLP representation in its exterior is set up by,

```
tref = segment.radialfunc([], {@(q) 1 + 0.3*cos(3*q), @(q) -0.9*sin(3*q),...
                    @(q) -2.7*cos(3*q)});   % note R" also given, helps later
d = domain([], [], tref, -1); d.k = 10;     % exterior domain & wavenumber
d.addlayerpot(tref, 'D');                   % adds DLP on tref segment
```

The coefficients in `p.co` represent *values* of the double-layer density function at the quadrature points. In contrast to MFS bases, the segment's own quadrature points are always used. Therefore the number of degrees of freedom is the current value of `numel(tref.x)`, and when you update N for a `layerpot` you actually change the quadrature point number of the underlying segment.

We may repeat the exterior BVP convergence of Sec. 4 using the DLP,

```
f = @(z) besselh(0, d.k * abs(z-0.3-0.2i));  % known exterior field
tref.setbc(1, 'D', [], @(t) f(tref.Z(t)));   % its Dirichlet data
p = bvp(d);
Ns = 5:5:80; for i=1:numel(Ns)
  p.updateN(Ns(i)); p.solvecoeffs; N(i) = p.N;
  e(i) = f(2) - p.pointsolution(pointset(2));   % error at a point in d
end
figure; semilogy(N, abs(e), '+-'); xlabel('N'); ylabel('error in u(2)');
```

The result is the solid line in Fig. 9a. The condition number of the problem is $O(1)$, viz. `cond(p.A)` gives 69, as opposed to the ill-conditioned value 1.8×10^7 for MFS in Sec. 4. Note that errors can no longer be estimated by `p.bcresidualnorm` since it always around machine precision, 10^{-16}.

What actually happened above? Quite a lot! The BVP class needed to know how to evaluate the DLP on its own source segment, which requires two new ingredients:

1. the *jump relation* that states that $u^+ = (\frac{1}{2}I + D)\tau$ where τ is the density function and D the integral operator on the boundary with the double-layer kernel, and,

2. a good quadrature scheme to evaluate $D\tau$ at the quadrature points themselves is needed; by default for periodic quadrature[14] the Martensen–Kussmaul scheme recommended by Kress is used [6], and this requires $z''(t)$, hence the need for R'' in `radialfunc`.

The above scheme is called the *Nyström method* [6]. How did `MPSpack` know to evaluate u^+ rather than u^-? Only the + side of the segment is connected to the evaluation domain. Note that our choice of normalization is that the matrix `p.A` maps density values to field values multiplied by the square-root of quadrature weights. To recover the unweighted matrix and

[14] `MPSpack` also implements Kapur-Rokhlin schemes of order 2, 4, and 10, which for non-FMM-accelerated problems sich as this are inferior to Kress'. For non-periodic open segments, no special quadratures are currently implemented.

check its eigenvalues, which approximate well eigenvalues of $(\frac{1}{2}I+D)$, we use

```
figure; plot(eig(diag(1./p.sqrtwei)*p.A), '+'); axis equal
```

giving Fig. 9b.

This plot shows an eigenvalue alarmingly close to zero. Why? It is well-known that $\frac{1}{2}I + D$ goes singular when k^2 is a Dirichlet eigenvalue of the interior domain [6]. This leads to blow-up of the condition number, hence non-robustness. A cure due to Brakhage, Werner, Leis and Panich [6] is to use a combined-field representation of DLP minus ik times SLP, which is done by replacing the above basis setup by

```
d.addlayerpot(tref, [-1i*d.k 1]);
```

Delightfully, the spectrum of $\frac{1}{2}I+D-ikS$ avoids zero like the plague (Fig. 9c) and now `cond(p.A)` is reduced to 5.

7.1 Transmission problems

We return now to the transmission problem from the end of Section 6. The Rokhlin–Müller scheme [10] for scattering from a dielectric obstacle uses independent single- and double-layer densities on the obstacle boundary curve, each of which is used to represent the field in both interior and exterior. This means that a single density must affect *two* domains in which, however, the wavenumbers are different. This is easy to set-up in `MPSpack`, and it contrasts the cases considered until now where a basis object may affect only a single domain. The advantage of this scheme is that the hypersingular kernels on the plus and minus sides of the surface cancel to give a second-kind Fredholm system. Some code for the below is found in `examples/dielscatrokh.m`

We create interior and exterior domains from a single segment (in fact the same one as in Sec. 4, using a simpler command), and enforce continuity conditions at their interface, as follows,

```
M = 330; s = segment.smoothstar(M, 0.3, 3);       % smooth closed segment
di = domain(s, 1); di.setrefractiveindex(n);      % interior
de = domain([], [], s, -1);                       % exterior
setmatch(s, 'diel', 'TM');
```

We use a new command to add 'double-sided' layer-potentials to the segment, one DLP and one SLP,

```
s.addinoutlayerpots('d'); s.addinoutlayerpots('s');
```

We may check that the DLP `di.bas{1}` affects both domains by checking that `di.bas{1}.doms` is a list of two domains. By default, a Kress

quadrature scheme is used for closed segments—this can be changed in the options to the above calls.[15]. Given the above domains (and once boundary/matching conditions and basis sets have been chosen), we set up then solve the problem with the same parameters as in Section 6,

```
p = scattering(de, di);
p.setoverallwavenumber(30);
p.setincidentwave(pi/6);              % if just angle given, it's a plane wave
p.solvecoeffs;
```

A call to `p.showthreefields` now reproduces Fig. 8. Note that in order to get 13 digits of accuracy at this medium wavenumber, we needed $M = 330$, i.e. 660 degrees of freedom. (This is larger than the number needed in Section 6, although the accuracy was lower there). At $k = 8$, $M = 110$ is sufficient for 14 digit accuracy. However, the linear system condition number `cond(p.A)` is only 6×10^3, enabling the use of iterative methods such as GMRES, in contrast to Sec. 6 in which it was highly ill-conditioned. The convergence rate is exponential.

In a future edition we will discuss sound-hard scattering and direct formulations. Please get in touch with the authors if you have any questions or suggestions!

8 Scattering from a periodic grating of obstacles

`MPSpack` includes recently-developed methods to periodize a scattering problem using layer-potentials on the unit-cell walls [2]. This means solving for the scattered field due to a *quasi-periodic* incident wave, such as a plane wave, from an infinite grating (one-dimensional array) of obstacles. Here we explain briefly how to set up and solve such problems, by building on layer-potential methods which solve the scattering problem for a single obstacle.

We will build on the transmission scattering case of Sec. 7.1. We first set up many elements of the single-obstacle scattering problem: we define a curve with sufficient number of numerical quadrature points on it to define interior and exterior domains, add layer-potential bases to them, and set the surface matching conditions,

```
s = scale(segment.smoothstar(80, 0.3, 3), 0.35);    % smooth closed curve
de = domain([], [], s, -1);                         % obstacle's exterior
```

[15] currently, since they cancel, the hypersingular parts of the T operator are not even included in the T evaluation

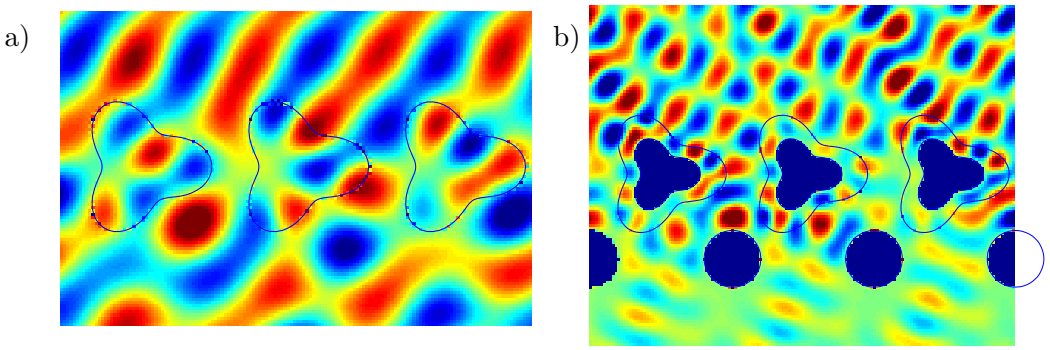

Figure 10: a) Plane wave diffraction from an infinite one-dimensional grating of transmission (dielectric) obstacles, solved with the `qpscatt` problem class as in Sec. 8. The real part of the full field is shown. The command `p.showbdry(struct('arrow',0,'normals',0,'label',0,'blobs',0));` was used to overlay the boundaries of the central three scatterers plotted as simple lines. b) Lattice of Dirichlet obstacles coated with dielectric layer of index 1.5, with nearby Dirichlet discs which wrap around the unit cell wall, plotted in the same way.

```
di = domain(s, 1); di.setrefractiveindex(1.5);      % interior, refractive index
s.addinoutlayerpots('d'); s.addinoutlayerpots('s'); % add Rokhlin LP scheme
s.setmatch('diel', 'TM');                           % TM dielectric continuity
```

Now comes the periodic ingredients of the problem, in particular the options parameters defining the Fourier-transform wall layer-potential representation. Here we give typical parameters which are good for low-wavenumber problems ($k < 30$ for a unit period). [16] These parameters are passed in as options when setting up the quasi-periodic problem, as follows,

```
d = 1.0;                                           % problem period
o.nei = 2; o.buf = 1; o.M = 150;                    % FTy LP method params
p = qpscatt(de, di, d, o);                          % set up the problem
```

The unit period means that the vertical unit-cell walls will lie at $x = \pm 1/2$. We may then choose an incident field (which sets the quasi-periodic Bloch phase, hence determines other method parameters), and solve and plot,

```
p.setoverallwavenumber(10);                         % incident wavenumber
p.setincidentwave(-pi/5);                           % incident plane wave angle
p.solvecoeffs;                                      % fill matrices and solve
p.showfullfield(struct('ymax', 1));                 % plot Re part of full field
```

The call to `p.solvecoeffs` took 0.5 sec, and computing the field for plotting took another 3.3 sec, which gives Fig. 10a. Three periods of the quasi-periodic full field are shown (one is computed and the others are phased copies of this one, so that one may check by eye that the quasi-periodicity condition is satisfied if the field shown appears smooth). The option `ymax` controls the upper and lower y-axis limits of the plotting domain; if this omitted, a sensible default is used which contains the obstacle and a small vertical border.

As with the non-periodic scattering problem, the total field values at points may now be found, e.g. at a single point via

```
p.pointsolution(pointset(0.3+0.7i))
```

This takes 0.06 sec (there is some overhead in handling the objects in the MATLAB implementation, which is dominant only when a small number of points is needed). Points lying outside the unit cell containing the origin[17]

[16] `nei` is the number of direct neighbor copies to sum, `buf` the extra buffer size of the quasi-periodic strip domain, and `M` half the number of degrees of freedom needed for periodizing.

[17] Or, when `buf>0`, the contiguous set of `1+2*buf` unit cells centered at the origin.

will not give correct answers, so in order to evaluate at such points the user should fold the points back into this cell and multiply by the appropriate Bloch phase. It is easy to extract the fractions of incident power that is reflected into each of the propagating Bragg diffraction grating orders,

```
[u d n] = p.braggpowerfracs(struct('table',1));     % intensity of Bragg orders
```

The sum of the flux fractions should be unity; the error from this is printed by the above, indicating an error of about 10^{-14}. A plot of vectors showing incident and Bragg directions may also be overlaid on the current field plot with `p.showbragg`; try it and see!

We conclude with a more complicated example, to show that this is also easy to set up in `MPSpack`. We consider transmission obstacles as above, but now containing a Dirichlet obstacle inside, and an isolated nearby Dirichlet disc-shaped obstacle. This is set up from scratch as follows,

```
d = 1.0; s = scale(segment.smoothstar(130, 0.3, 3), 0.35);
sm = s.scale(0.6); di = domain(s, 1, sm, -1);  % dielectric coating domain
di.setrefractiveindex(1.5);
c = segment(60, [.5-.6i, .2 0 2*pi]);          % small circle segment
de = domain([], [], {s c}, {-1 -1});           % twice-punctured exterior domain
s.addinoutlayerpots('d'); s.addinoutlayerpots('s');
om = 20;                                       % incident wavenumber
di.addlayerpot(sm, [-1i*om 1]); de.addlayerpot(c, [-1i*om 1]);         % CFIEs
s.setmatch('diel', 'TM'); sm.setbc(1, 'D', []); c.setbc(1, 'D', []);   % BCs
o.nei = 2; o.buf = 1; o.M = 150; p = qpscatt(de, di, d, o);    % set up problem
```

In addition, we now have a higher wavenumber, and we choose an incident wave direction corresponding to a Wood's anomaly [9]. The solution is completed by,

```
p.setoverallwavenumber(om); p.setincidentwave(-acos(1-2*pi/om)); % Wood's anom
p.solvecoeffs; p.showfullfield(struct('ymax', 1.2));
```

which produces Fig. 10b. The numbers of quadrature nodes have been chosen to give 13 digits of accuracy, giving `p.N` of 450 obstacle degrees of freedom (751 total degrees of freedom including the quasi-periodizing scheme). The solution time is 2 sec, the plotting time 7 sec.

For those interested in the solution method, the Sommerfeld contour and poles for the Fourier-transform layer-potentials can be visualized via `p.showkyplane`; see [2].

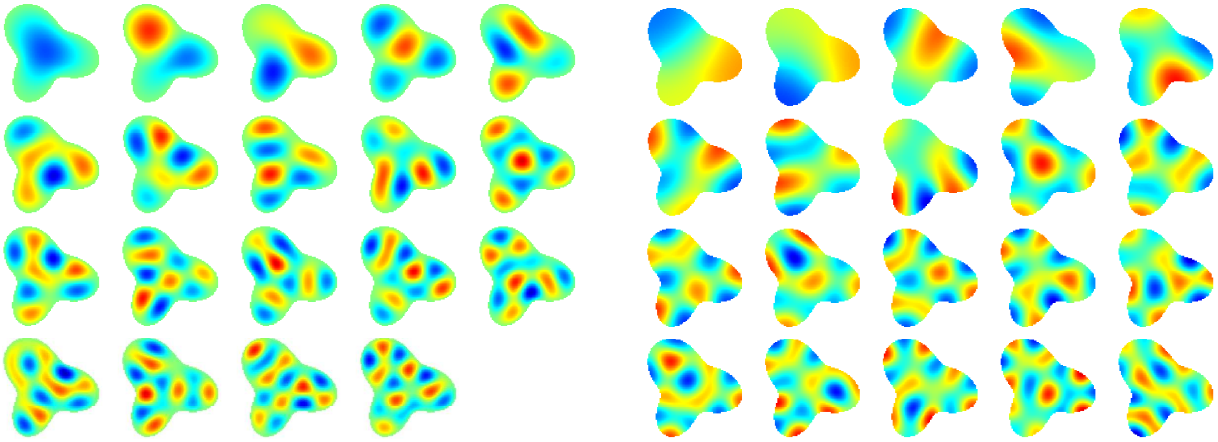

Figure 11: The lowest few Dirichlet (left) and non-trivial Neumann (right) eigenmodes for the smooth nonsymmetric domain eigenvalue problem.

9 Dirichlet and Neumann eigenvalue problems

We now illustrate the solution of some Laplace eigenvalue problems of the form

$$-\Delta u = k^2 u \qquad \text{in } \Omega$$

where Ω is a bounded domain, with homogeneous boundary conditions either

$$u = 0 \qquad \text{on } \partial\Omega \qquad \text{(Dirichet)}$$

or

$$u_n = 0 \qquad \text{on } \partial\Omega \qquad \text{(Neumann)} \ .$$

This has a discrete set of non-trivial solutions $u = \phi_j$ (eigenmodes) and corresponding $k = k_j$ (eigenfrequencies), labeled in nondecreasing frequency order by $j = 1, 2, \ldots$. We start with a smooth domain, which (unlike the trefoil domain used earlier) is non-symmetric. We apply Dirichlet boundary conditions and set up an eigenvalue problem (EVP) object.

```
s = segment.smoothnonsym(80, 0.3, 0.2, 3);
d = domain(s, 1);          % create an interior domain
s.setbc(-1, 'D');          % Dirichlet BC's applied on inside of segment
p = evp(d);                % sets up eigenvalue problem object
```

There are three main solver methods coded into `MPSpack`, all of which are based on layer potentials (boundary integral operators):

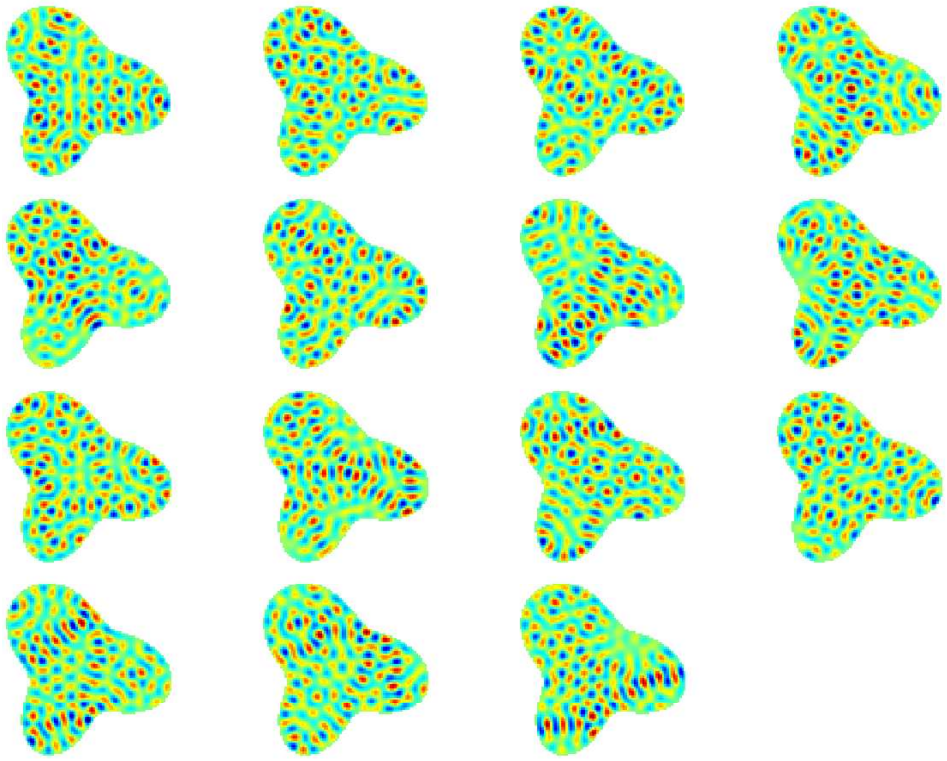

Figure 12: All Dirichlet eigenmodes with eigenfrequencies lying in $[30, 31]$ for the smooth nonsymmetric domain eigenvalue problem.

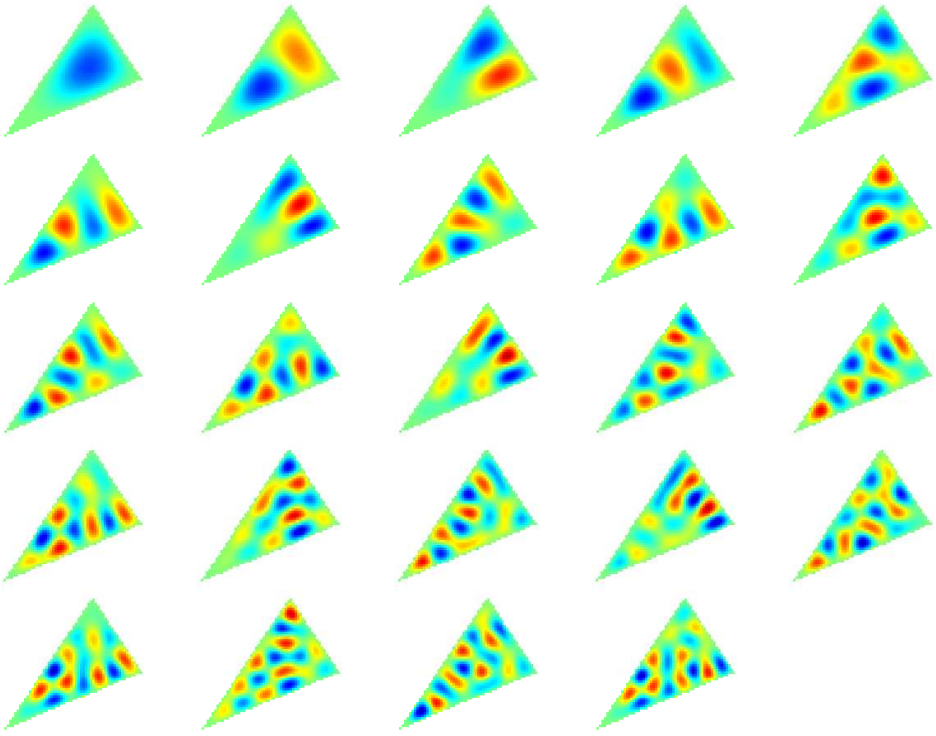

Figure 13: The lowest Dirichlet eigenmodes of a triangle with $k_j \leq 20$, computed in 7 seconds to 9 digit accuracy via the Fredholm determinant method.

- `'fd'`: the Fredholm determinant method, most efficient and convenient for low-lying eigenvalues (i.e. the lowest few hundred).
- `'ntd'`: the Neumann-to-Dirichlet map method [3], most efficient for high-lying Dirichlet eigenvalues (not available for Neumann case).
- `'ms'`: the minimum singular-value method. This is less efficient (around 10 times slower than `fd`), but serves as an accurate and reliable reference method (it handles degeneracies better than `fd`).

To use the first method a double-layer potential basis is needed for Dirichlet (single-layer would be used for Neumann). Then to solve for all eigenvalues lying in $2 \leq k_j \leq 10$ the call is as follows.

```
d.addlayerpot(s, 'd');          % basis set appropriate for BCs
p.solvespectrum([2 10], 'fd');  % Fredholm det method
```

In 3 seconds, 19 eigenvalues are found and stored in `p.kj`. Their accuracy may be estimated by `p.ej` which shows around 13 digits at the low end, deteriorating slightly to 9 digits at the high end: this is due to the choice of $N = 80$ which would need to be increased to handle higher frequencies. A simple (but crude and not completely rigorous) check whether any eigenvalues were missed is via Weyl's law:

```
p.weylcountcheck(p.kj(1),p.kj,d.perim,d.area);
```

The resulting graph should oscillate with a range of around 1 without making any permanent jumps of integer sizes; such jumps up indicate missing (or if down, duplicated) eigenvalues. This is most useful for testing large numbers of eigenvalues.

No information about the modes ϕ_j was stored. To also compute modes, and then to plot them, use the following.

```
o.modes = 1; p.solvespectrum([2 10], 'fd', o);
p.showmodes;
```

The solution takes 4 sec, and the plotting 2 sec. Mode boundary functions $\partial_n \phi_j$ are returned in the array `p.ndj` from the right singular vector of $(I - 2D^*)$, and the double-layer density which generates them in `p.coj` from the left singular vector. The plotting is done by default by evaluation of the single-layer part of the Green's representation formula for ϕ_j, using the normal-derivative data, since this doesn't involve any double-layer evaluation. The result is the left side of Fig. 11. When modes are requested, the

minimum singular values are also checked for the eigenfrequencies, giving an independent test of the error. Errors in modes depends on the separation between eigenvalues, as in matrix eigenvalue problems, and would be better assessed by a convergence study in N. The mode data on a grid can be accessed via `[uj gx gy di js] = p.showmodes;` The mode is L^2-normalized on Ω.

For degenerate eigenvalues you will need to set the option `o.iter = 0` which switches to the usual SVD for mode checking; this is slower but scales with N the same way, $O(N^3)$. This is in development.

The Neumann case (switching to `'N'` in `setbc` and `'s'` in `addlayerpot`), and solving in $1 \leq k_j \leq 7.8$ gives the right side of Fig. 11. As before, the eigevalue estimated accuracies vary from 15 to 10 digits. Note that the lowest k in this range cannot include zero; of course the Neumann case does have a trivial eigenpair $k_0 = 0$, $\phi_0 \equiv 1$. Mode evaluation is by default done by evaluating the single-layer potential given by the left singular vector of $(I + 2D)$, since this doesn't involve any double-layer evaluation. The mode is approximately L^2-normalized on Ω, using the evaluation grid.

When the domain has symmetries, the `'fd'` method is not so good, and does not find modes spanning the eigenspace. Switching to `'ms'` or `'ntd'` is more reliable.

To illustrate computing high-lying Dirichlet eigenvalues as in [3], set up the Dirichlet case as above then try the following.

```
p.updateN(300);
o.eps = 0.1; o.khat = 'r'; o.fhat = 's'; o.modes = 1;
p.solvespectrum([30 31], 'ntd', o);
p.showmodes;
```

This takes 4 sec to solve and 4 sec to plot, producing Fig. 12. For higher-frequencies, fast multipole evaluation of the modes is much more efficient. This is not yet documented.

In order to handle corner domain eigenvalue problems, one needs to use special quadratures, for instance for a triangle use

```
o.kressq = 5;        % corner-packing parameter for Kress reparametrization
s = segment.polyseglist(50, [1, exp(3i*pi/8), exp(5i*pi/4)], 'pc', o);
```

The result for $k_j \in [4, 20]$ is Fig. 13. The accuracy of this is currently limited to around 10 digits, because increasing `kressq` can result in catastrophic cancellation due to geometry representation near the corner; this needs to be fixed. To stop less-accurate modes from being dropped by the solver,

set an option such as `o.tol = 1e-4`. A more complicated domain is also illustrated in Section 10.4. Neither `'ntd'` nor `'ms'` works with corners yet in `MPSpack`. See `verg` code on the first author's webpage for the former.

The reader may benefit from also seeing the codes `test/testevp.m`, `test/testevpms.m` and the other codes in `examples/` .

9.1 Method of Particular Solutions and inclusion bounds

The MPS [5] uses basis functions other than layer potentials, and therefore results in ill-conditioned and/or rectangular matrices. Ironically, despite the name of this package, code for the MPS approach to eigenproblems is in a less complete state.

New high-frequency Neumann inclusion bounds (Barnett–Hassell–Tacy 2015) are implemented in the codes in `examples/neumann_inclusion`; see the `README`. This produces for example, Fig. 14, with 15 digits of eigenvalue accuracy. This is computed by the code `examples/neumann_inclusion/tbl_mfsgsvdincl.m`. An MFS basis is used, with imaginary displacement for the source point choice.

Work is needed to bring in the various Dirichlet methods into the `evp.gsvdtension.m` code which is the optimal implementation for the Neumann case.

There are also MPS methods such as `evp.crudempssolvespectrum` that are incomplete, orphaned, or in development.

We have not tested `MPSpack` for mixed D-N boundary conditions, although it is probably quite easy to set that up by editing some of the codes in `@evp`.

10 More elaborate examples

In this section we describe in detail some more complicated worked examples and give the corresponding code. All examples can also be found in the `examples` subdirectory of `MPSpack`.

10.1 Scattering from an array of discs

In this section we describe how to solve the problem of scattering from an array of circular scatterers in `MPSpack` using a basis of fundamental solutions.

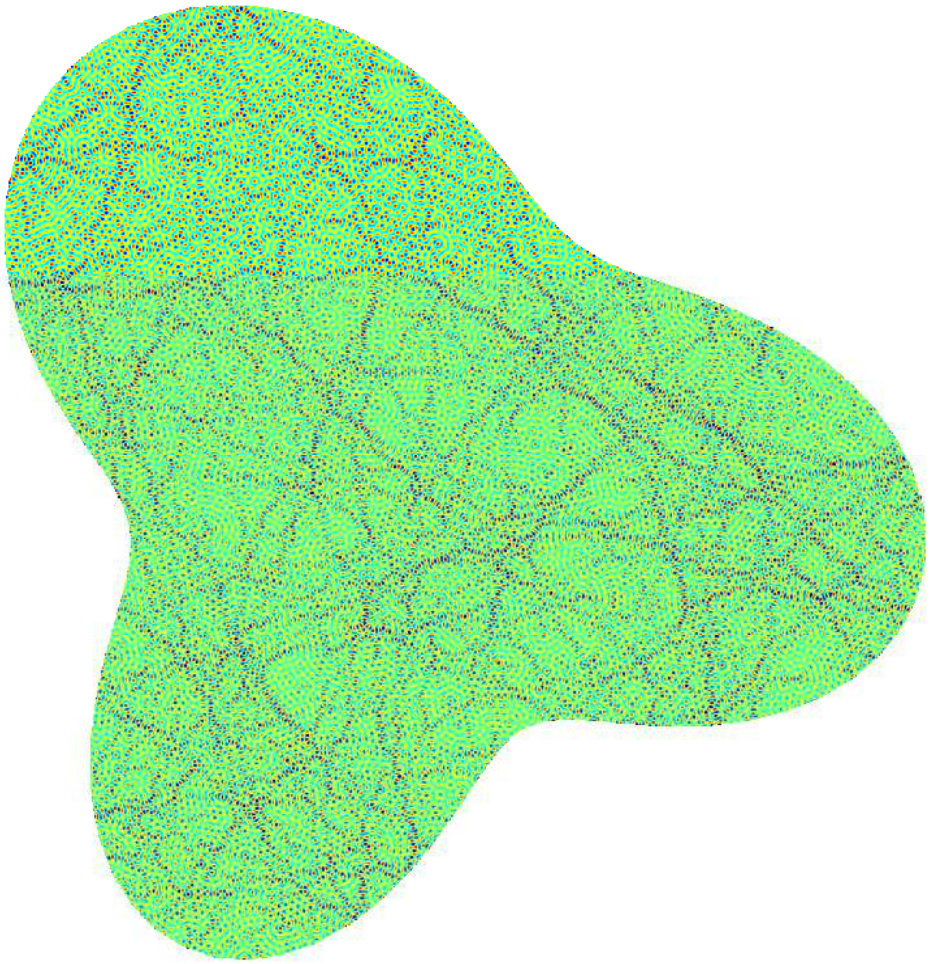

Figure 14: Neumann eigenmode of a nonsymmetric smooth planar domain `s = segment.smoothnonsym(N, 0.3, 0.2, 3);` with eigenfrequency $k = 405.003269518228\ldots$, around the 42612th eigenvalue of this domain.

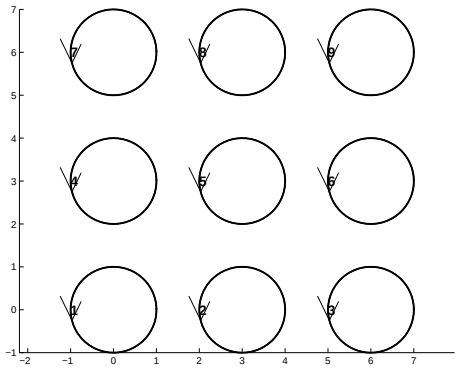

Figure 15: An array of 9 scatterers.

10.1.1 Problem description and solution approach

Let Ω be the union of disks D_j with radius r whose midpoints have the coordinates $(an_1^{(j)}, an_2^{(j)}) \in \mathbb{R}^2$, where $n_1^{(j)}, n_2^{(j)} \in \mathbb{N}$ and $a > 0$ is the distance between two neighboring midpoints. An example for 9 scatterers is shown in Figure 15. The PDE is the following.

$$\begin{aligned} \Delta u + k^2 u &= 0 \quad \text{in } \mathbb{R}^2 \backslash \Omega && (1) \\ u &= 0 \quad \text{on } \partial\Omega && (2) \\ \frac{\partial u_s}{\partial r} - iku_s &= o(r^{-1/2}), && (3) \end{aligned}$$

Here, $u = u_{inc} + u_s$ is the total field, u_{inc} is the incident wave, u_s is the scattered field, and r is the radial coordinate. The Sommerfeld radiation condition (6) is to be understood to hold uniformly in all directions.

We solve the problem using fundamental solutions approximations in each disk, where the charge points lie on circles with radius $r_{mfs} < r$.

10.1.2 Implementation in `MPSpack`

Initialization of the problem parameters We have the following problem parameters.

```
N1=3; N2=3; % Number of scatterers in each direction
r=1; % Radii of circles
a=3; % Distance of midpoints of neighboring circles in each dimension
k=10; % Wavenumber
```

```
M=300; % Number of points on each circle
N=150;  % Number of MFS basis fct. in each circle
Rmfs=0.8*r; % Radius of fundamental solutions inside circles
```

Setup of the geometry The geometry is setup using a simple for loop.

```
y0=0;
s=segment.empty(N1*N2,0);
for i=1:N1,
    x0=0;
    for j=1:N2
        seg=segment(M,[x0+1i*y0 r 0 2*pi],'p');
        seg.setbc(1,'D',[]);
        s((i-1)*N2+j)=seg;
        x0=x0+a;
    end
    y0=y0+a;
end
```

In the inner for loop circular segments are created whose handle is the variable `seg`. We directly assign the homogeneous Dirichlet boundary conditions using the command

```
seg.setbc(1,'D',[]);
```

The segments are then stored in the array `s`. Storing the circular segments in an array makes plotting easy. Figure 15 is simply created with the commands

```
o.normals=0;
plot(s,1,o);
```

Once all segments are created we define the domain by

```
d=domain([],[],num2cell(s),num2cell(-1*ones(N1*N2,1)));
```

This command might look complicated at first because of the use of `num2cell`. We need to convert the array `s` into a cell structure so that the `domain` constructor recognizes that each element of s is a closed shape in itself. Otherwise, the constructor would try to combine the elements of `s` into one connected shape, which is not possible.

Basis functions We want to use a basis of fundamental solutions in each of the circular scatterers. This is achieved with the following for loop.

```
x0=0; y0=0; opts.fast=1;
for i=1:N1,
    x0=0;
    for j=1:N2
        Z=@(w) Rmfs*exp(2i*pi*w)+x0+1i*y0;
        Zp=@(w) Rmfs*2i*pi*exp(2i*pi*w);
        d.addmfsbasis({Z,Zp},N,opts);
        x0=x0+a;
    end
    y0=y0+a;
end
```

The function handle `Z` defines the fundamental solutions curve and `Zp` its derivative. We have to shift the curves according to the midpoints of the circles.

Solving the problem We now have everything together to setup a problem instance. This is easily done with the `scattering` class and the following three commands.

```
pr=scattering(d,[]);
pr.setoverallwavenumber(k);
pr.setincidentwave(-pi/3);
```

The solution and timing results are then obtained by

```
tic; pr.solvecoeffs; fprintf('\tcoeffs done in %.2g sec\n', toc)
fprintf('\tL2 bdry error norm = %g, coeff norm = %g\n', ...
        pr.bcresidualnorm, norm(pr.co));
```

On a standard Core 2 Duo Laptop this takes just under 40 seconds with a boundary error of about $2 \cdot 10^{-13}$. We can plot the solution using the following commands.

```
o.bb=[-a N2*a -a N1*a];
o.dx=0.05;
o.sepfigs=1;
pr.showthreefields(o);
```

The command `pr.showthreefields(o)` plots the incoming wave, scattered field and full field using the parameters given in `o`. We have specified

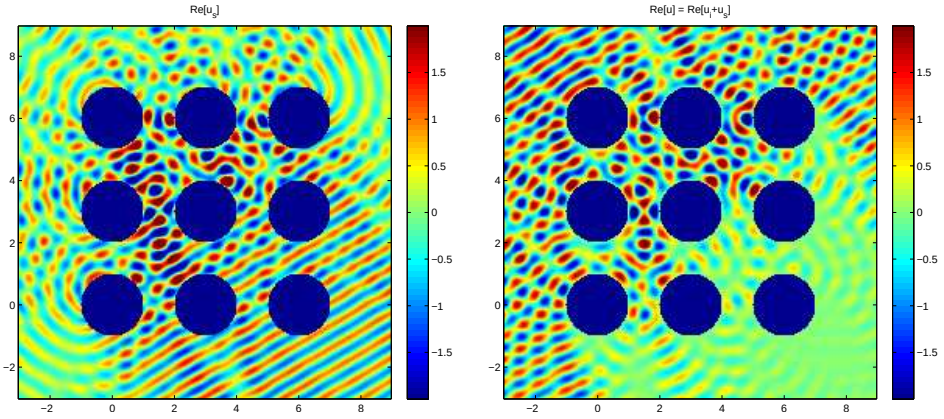

Figure 16: The scattered (left) and the full field (right) for the problem of scattering from an array of circles.

`o.sepfigs=1`. This creates three separate figures for the three different fields instead of plotting everything into one figure.

The scattered and the full field are shown in Figure 16. Note that plotting can take a long time since the array of scatterers spans many wavelenghts for which we need a sufficiently high resolution. The solution is not symmetric with respect to the diagonal since the incident angle for the incoming wave is $-\frac{\pi}{3}$, destroying the symmetry given by the configuration of scatterers.

10.2 Transmission scattering with 'metal' and air holes

Here we demonstrate how the fundamental solutions introduced in Sec. 4 can be used to solve efficiently the scattering of TE (transverse-electric) polarized z-invariant plane-wave (2D Maxwell equations) from a smooth dielectric body with various inclusions. We will be somewhat brief in this example.

The mathematical set up is identical to the previous example, except that there is a bounded dielectric region which has refractive index $n = 1.5$ and hence a Helmholtz solution with larger wavenumber than in the surrounding vacuum (or 'air'). The field u represents H_z, the out-of-plane magnetic field. The solution in the dielectric must match that in the air at their common boundary, as we describe shortly. We also have a homogeneous boundary condition on a 'metallic' inclusion inside the dielectric; see Fig. 17.

First we set up some curves, formed by scaling, translating and rotating

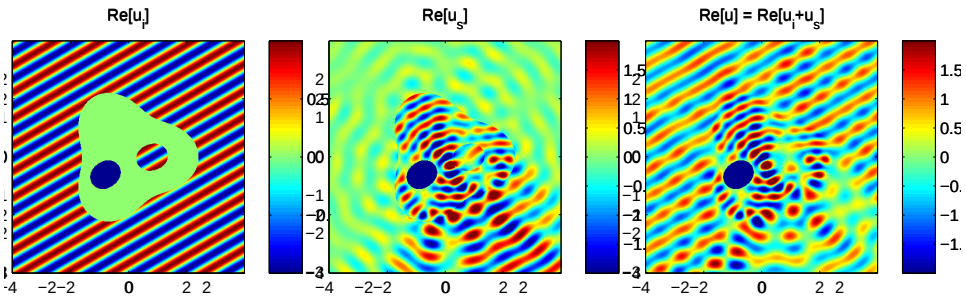

Figure 17: Transmission scattering (TE-polarized) from dielectric with two inclusions: an air hole (to the right side) and a 'metallic' PMC inclusion (to the left side). Multiple MFS basis curves are used. Left: incident wave u_{inc}. Center: scattered wave u_s. Right: their sum, the total solution field u.

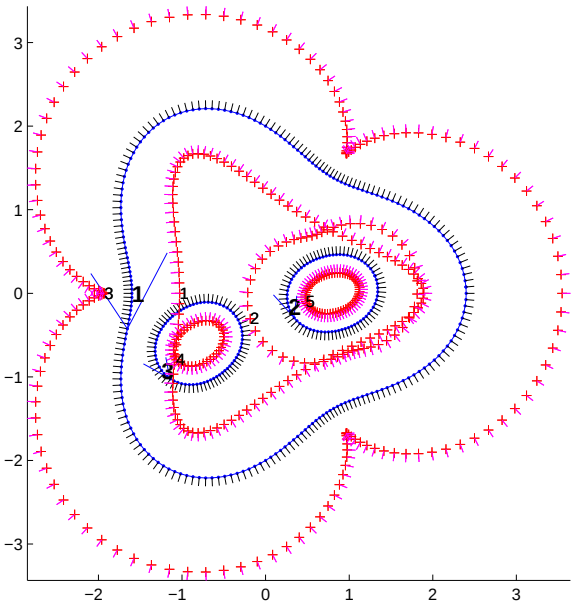

Figure 18: MFS curves (red) and boundary curves (blue) used for transmission scattering from dielectric with two inclusions.

standard 'radius-1' segments `'s'` and `'c'`, and build the three domains from them,

```
s = segment.smoothstar(200, 0.2, 3);          % weak trefoil shape
c = segment.smoothstar(70, 0.1, 2);           % squashed circle

sd = s.scale(2);                              % outer bdry of dielectric
sa = translate(rotate(sm,.3), .8);            % air inclusion bdry
sm = c.scale(0.5); sm.rotate(pi/5); sm.translate(-.8-.6i); % 'metal' bdry

de = domain([], [], sd, -1);                  % exterior air domain
da = domain(sa, 1);                           % air inclusion domain
d = domain(sd, 1, {sm sa}, {-1 -1});          % dielectric w/ 2 holes
d.setrefractiveindex(1.5);                    % choose dielectric index
```

Assuming relative permeability of unity, the relative permittivity of the dielectric is $\epsilon = n^2$. The TE boundary condition is that u is continuous across a dielectric interface (segments `sd` and `sa`), while for the normal derivatives $\epsilon^{-1}\partial u/\partial n$ is continuous.[18] We set this up, and a Dirichlet BC on the other inclusion, with,

```
setmatch([sd sa], 'diel', 'TE');        % impose TE matching diel-air
sm.setbc(1, 'D', []);                   % Dirichlet (PMC) BC on 'metal'
```

The domains are then collected into a scattering problem as follows (note that by including `da` in the first argument list, it is set up as an 'air' domain thus receives a nonzero u_{inc} plane-wave field, although results are similar if `de` is the only air domain),

```
pr = scattering([de da], d);   % includes air pocket in nonzero u_inc
pr.setoverallwavenumber(8);    % incident (air) wavenumber
pr.setincidentwave(-pi/3);     % if just angle given, assumed plane wave
```

MFS bases are very convenient for such analytic boundaries, and we use the analytic continuation of the boundary parametrization to generate MFS curves either inside or outside the actual segment curves (`tau>0` is inside, and `tau<0` outside, for CCW segments). Fig. 18 shows the five MFS curves and the three segments. The reason the dielectric needs three MFS curves is that (analogous with Runge's Theorem in complex approximation), singularities are needed in *each* disconnected component of the domain's complement. The distances `tau` in the following should be chosen as large as possible

[18] See [8]. See `segment.dielectriccoeffs` for where this is set up in `MPSpack`.

while still keeping the coefficient norm `norm(pr.co)` not too large (i.e. 10^4 or less).

```
de.addmfsbasis(sd, 120,struct('tau',0.05));  % exterior domain
da.addmfsbasis(sa, 60, struct('tau',-0.1));  % inside air pocket
d.addmfsbasis(sd,  150,struct('tau',-0.05)); % 1st of 3 curves for diel...
d.addmfsbasis(sm,  60, struct('tau',0.1));   % ...2nd curve in 'metal' hole
d.addmfsbasis(sa,  60, struct('tau',0.1));   % ...3rd curve in air hole
```

Note that the basis set degrees were optimized roughly for this k value, giving only 450 total degrees of freedom for a problem around 8 wavelengths in size. Solving for the coefficients of the solution takes 0.67 sec,

```
pr.solvecoeffs; pr.bcresidualnorm
```

The L^2 boundary error is less than 10^{-7}. We then plot the incident, scattered, and total field as usual with `pr.showthreefields;` giving Fig. 17. Evaluating the solution on a 400×400 grid takes 11 sec.

10.2.1 Convergence study

Once a good set of MFS basis sizes has been chosen, as above, they may all be changed at once conveniently with the `nmultiplier` option. We could have redone the basis sets as follows,

```
d.clearbases; de.clearbases; da.clearbases;
de.addmfsbasis(sd, [], struct('tau',0.05, 'nmultiplier', 120/450));
da.addmfsbasis(sa, [], struct('tau',-0.1, 'nmultiplier', 60/450));
d.addmfsbasis(sd, [],  struct('tau',-0.05,'nmultiplier', 150/450));
d.addmfsbasis(sm, [],  struct('tau',0.1,  'nmultiplier', 60/450));
d.addmfsbasis(sa, [],  struct('tau',0.1,  'nmultiplier', 60/450));
```

The mutipliers were set up to be the fraction of the total degrees of freedom that they were before, but now they can be scaled with a single command, viz, `pr.updateN(450)` sets up the original basis sizes, as may be checked with `diff([pr.basnoff pr.N])` which gives the numbers of degrees of freedom in each problem basis set. We put this in a convergence loop,

```
Ns=300:50:600; for i=1:numel(Ns)
  pr.updateN(Ns(i)); pr.solvecoeffs; u0 = pr.pointsolution(pointset(0));
  fprintf('N=%d, co-norm=%g, bc-norm=%g, |u(0)|=%.16g\n', Ns(i), ...
          norm(pr.co), pr.bcresidualnorm, abs(u0));
end
```

We actually doubled the number of quadrature points on the original segments to perform this high-accuracy study, giving 680 quadrature points in total. The output shows very rapid convergence,

```
N=300, co-norm=2125.38, bc-norm=0.000363003, |u(0)|=3.045222448551815
N=350, co-norm=1968.37, bc-norm=1.26165e-05, |u(0)|=3.045222451567895
N=400, co-norm=1844.05, bc-norm=4.21908e-07, |u(0)|=3.045222451435764
N=450, co-norm=1746.25, bc-norm=9.77848e-08, |u(0)|=3.045222451435782
N=500, co-norm=1651.79, bc-norm=3.96167e-09, |u(0)|=3.045222451436035
N=550, co-norm=1581.87, bc-norm=1.1487e-09,  |u(0)|=3.045222451436006
N=600, co-norm=1521.16, bc-norm=1.44025e-10, |u(0)|=3.045222451436037
```

This suggests that, at least for u at the origin, we had 12 correct digits in the solution with the total $N = 450$ used above, and probably achieved 14 digits with $N = 600$, computed in just a couple of seconds.

10.3 Acoustic scattering from the unit square

This example introduces a new feature: the use of decomposition of a region of constant wavenumber into computational subdomains connected by fictitious boundaries. The code is in `examples/tut_square.m`

10.3.1 Problem description and solution approach

In this example we solve the problem of time-harmonic acoustic sound-soft scattering from the unit square $\Omega = (-0.5, 0.5)^2$. The full PDE has the following form.

$$\Delta u + k^2 u = 0 \quad \text{in } \mathbb{R}^2 \backslash \Omega \tag{4}$$

$$u = 0 \quad \text{on } \partial\Omega \tag{5}$$

$$\frac{\partial u_s}{\partial r} - iku_s = o(r^{-1/2}), \tag{6}$$

Here, $u = u_{inc} + u_s$ is the total field, u_{inc} is the incident wave, u_s is the scattered field, and r is the radial coordinate. The Sommerfeld radiation condition (6) is to be understood to hold uniformly in all directions.

To achieve high accuracy we cannot simply use fundamental solutions to approximate the scattered field. The problem is the singularities of the solution u at the corners. If these are not represented in the basis our approximation error will decay very slowly as the number of basis functions increases. To solve this problem we use the domain decompositon shown in Figure 19. The idea is that each of the elements E_i only contains one corner

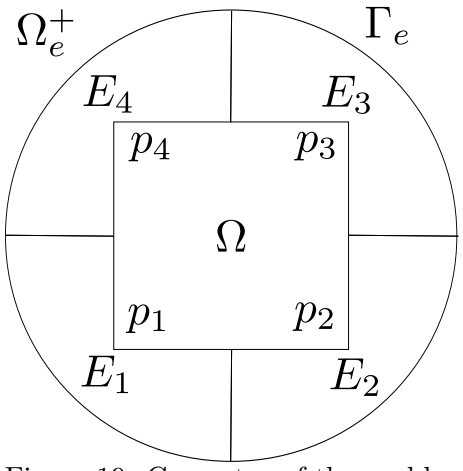

Figure 19: Geometry of the problem

p_i of the square. We can then match the corner behavior in each domain by using fractional order Bessel functions. Since furthermore, $u = 0$ on $\partial\Omega$ it will be sufficient to use Fourier-Bessel sine functions that automatically satisfy the zero boundary conditions on the sides of the square. Hence, the total field u is approximated in each element E_i using an approximation of the form

$$u(r,\theta) \approx \sum_{j=1}^{N_i} c_j^{(i)} J_{\frac{2}{3}j}(kr)\sin(\frac{2j}{3}\theta).$$

The polar coordinate system in each element E_i is rotated in such a way that the basis functions are zero on the sides adjacent to the corner at p_i.

In the infinite domain Ω_e^+ we use a basis of fundamental solutions to represent the scattered field u_s. Hence, for $\mathbf{x} \in \Omega^+$ we have

$$u_s(\mathbf{x}) \approx \sum_{j=1}^{N_e} \frac{i}{4} c_j^{(e)} H_0^{(1)}(|\mathbf{x} - \mathbf{y}_j|),$$

where $\mathbf{y}_j = r_{mfs} e^{i\phi_j}$ and $\phi_j = \frac{2\pi j}{N_e}$. Note that we approximate with the fundamental solutions the scattered field u_s while in the finite subdomains E_i we approximate the total field u. The compatibility conditions between approximations u^i und u^j in two neighboring elements E_i and E_j with common boundary Γ_{ij} are given by

$$u^{(i)}(\mathbf{x}) \approx u^{(j)}(\mathbf{x}), \quad \frac{\partial u}{\partial n_i} u^{(i)}(\mathbf{x}) \approx \frac{\partial u}{\partial n_i} u^{(j)}(\mathbf{x}),$$

where $\mathbf{x} \in \Gamma_{ij}$ and $\frac{\partial}{\partial \nu_i}$ is the outward normal derivative at the boundary of E_i. On the interface Γ_{ie} between an element E_i and the exterior domain Ω_e^+ we have the compatibility conditions

$$u^{(i)}(\mathbf{x}) \approx u_{inc}(\mathbf{x}) + u_s^{(e)}(\mathbf{x}), \quad \frac{\partial}{\partial n_i} u^{(i)}(\mathbf{x}) \approx \frac{\partial}{\partial n_i}(u_s^{(e)} + u_{inc})(\mathbf{x}),$$

where $u_s^{(i)}$ is the fundamental solutions approximation to the scattered field in Ω_e^+. We have to add the incident field to the approximate scattered field since we are matching with the total field in the elements E_i. An approximate solution to the whole problem is now computed by minimizing the L^2 error of the compatibility conditions on all interfaces.

10.3.2 Implementation in `MPSpack`

Although the setup of the problem seems quite complicated we will see that it is very simple to set it up in `MPSpack`. Indeed, the most part of the code will be devoted to creating the mesh structure from Figure 19.

Initialization of the problem parameters We need to define the following problem parameters.

```
k = 50;       % Wavenumber
r = 1.0;      % Radius of outer circle
M = 200;      % Number of quadrature points on segments
N=100;        % Number of basis fct. in each subdomain
a=.5;         % Half-Size of the square
rmfs=0.8*r;   % Radius of the fundamental solutions curve
```

Setup of the geometry We now need to define the geometry. Fortunately, `MPSpack` gives some support for the construction of the geometry.

The list of segments is defined by the following three commands.

```
s = segment.polyseglist(M, [1i*r 1i*a a+1i*a a r]);
s=[s(1:3) segment(2*M, [0 r 0 pi/2])];
s = [s rotate(s, pi/2) rotate(s, pi) rotate(s, 3*pi/2)];
```

The first command defines all the straight lines that form part of the boundary of E_3. For this we use `polyseglist`. The command polyseglist constructs a closed polygon. We then delete the last two segments of the array `s` and add instead the circular line segment. This results in the segments

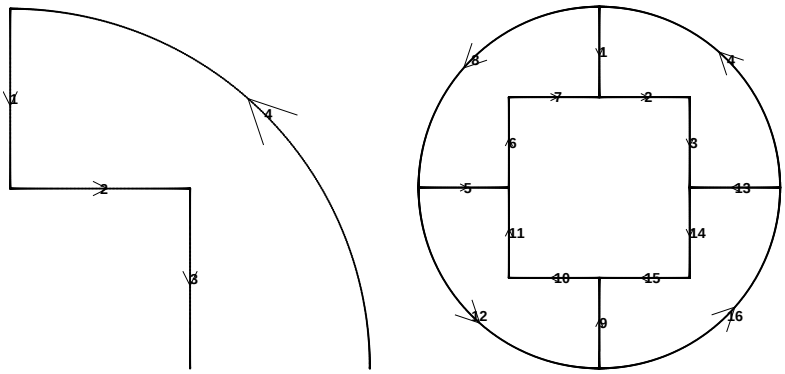

Figure 20: The segments in the right plot are created by rotating the segments shown in the left plot.

shown in the left plot of Figure 20. We now rotate this element three times to obtain the segments showin in the right plot of Figure 20.

For later it is important to have a separate list of all segments not belonging to the square and all segments belonging to the outer circle.

```
sdecomp=s([1 4 5 8 9 12 13 16]); % All artificial boundaries
extlist=s([4 8 12 16]);          % Segments forming the outer circle
```

We now define the domains. By taking the rotational symmetry into account we can do this in a simple for loop.

```
d=domain.empty(4,0);
for j=1:4, d(j)=domain(s(1+mod(4*(j-1)+[0 1 2 12 3],16)),[1 1 1 -1 1]); end
ext = domain([], [],extlist(end:-1:1), -1);
```

The for loop looks slightly complicated. But all it does is pick out the right indices for the elements forming a domain and creating it together with the right sense of direction. At the end we have an array `d` containing the four fine domains E_i. The exterior domain `ext` is created by traversing `extlist` in reverse order with reversed sense -1. This is necessary since we now have the boundary of an exterior domain, which has a reversed sense of direction.

Boundary conditions and basis functions Setting up the compatibility conditions between the elements is trivial. It is done by the command

```
sdecomp.setmatch([k -k],[1 -1]);
```

The matching conditions for the function values are scaled by the wavenumber k to balance the different scaling between the L^2 error in the function and the L^2 error in the derivative.[19]

We can now add the basis functions to the domains. The fractional Bessel functions are added to the interior domains by the following command.

```
nuopts=struct('type','s','cornermultipliers',[0 0 1 0 0],'rescale_rad',1);
for j=1:4, d(j).addcornerbases(N,nuopts); end
```

The options structure nuoopts specifies that we only want Fourier-Bessel sine functions at the third corner of each domain. This is the corner belonging to the square. The option 'rescale_rad' specifies that the basis functions are rescaled to balance out the bad scaling of Bessel functions. The method addcornerbases automatically finds out the right fractional orders, offsets and suitable branch vectors.

The exterior fundamental solutions are added with the following command.

```
Z=@(t) rmfs*exp(2i*pi*t); Zp=@(t) 2i*pi*rmfs*exp(2i*pi*t);
opts=struct('eta','k','fast',1,'nmultiplier',2);
ext.addmfsbasis({Z, Zp},N,opts);
```

Note that now nmultiplier is set to 2. It turns out to be effective for this problem to use twice as many fundamental solutions as there are Fourier-Bessel sine functions in each domain.

Solving the problem We now have everything together to setup the problem class and solve the scattering problem. The following commands setup the scattering problem and define an incident plane wave.

```
pr=scattering(ext,d);
pr.setoverallwavenumber(k);
pr.setincidentwave(-pi/4);
```

There is one small specialty here. In the first line we have defined ext to be an air-domain and the array d to be a non-air domain. This tells MPSpack to add the incident field to the exterior basis functions in assembling the least-squares problem. In the finite domains this is not necessary since there we approximate directly the full field.

The following commands now solve the problem and plot the solution u.

[19] Consider the one dimensional plane wave e^{ikx}. The derivative is ike^{ikx}. Hence, in general it makes sense to scale the L^2 error of function values by k to give it the same dimension as the L^2 error of the derivative.

```
tic; pr.solvecoeffs; fprintf('\tcoeffs done in %.2g sec\n', toc)
fprintf('\tL2 bdry error norm = %g, coeff norm = %g\n', ...
        pr.bcresidualnorm, norm(pr.co))
o.bb=[-1.5 1.5 -1.5 1.5];
o.dx=0.02;

[ui gx gy] = pr.gridincidentwave(o);
u = pr.gridsolution(o);

figure;
imagesc(gx, gy, real(ui+u)); title('Full Field (Real Part)');
c = caxis; caxis([-1 1]*max(c));
axis equal tight;
colorbar;
set(gca,'ydir','normal');
```

The incident field ui is automatically evaluated only in air-domains. If we want to evaluate it in all domains we have to set o.all=1. But here the default behavior is fine for us. The sum ui+u is now the total field in all domains (remember that u approximates the total field in the interior domains and the scattered field in the exterior domain). The scattering class also provides a routine showthreefields to plot the incident wave, scattered field and total field. However, the routine assumes that the computed solution is the scattered field, which is not correct for the way we have set up this problem. The resolution if the plot can easily be increased by decreasing the variable o.dx, which influences the distance of two grid points.

The output of the example problem is shown in Figure 21. The L^2 boundary error of the solution is approximately $1.5 \cdot 10^{-10}$. On a standard laptop with Intel Core 2 Duo processor the solution vector is computed in around 11 seconds. The plot takes slightly longer.

A wonderful feature of this approach is that we can trivially switch to a sound-hard scattering problem, that is instead of requiring $u = 0$ on $\partial\Omega$ we require $\frac{\partial}{\partial n}u = 0$ on $\partial\Omega$. All we have to do is switch to Fourier-Bessel cosine functions. These automatically satisfy the required condition for the normal derivative. The solution to this problem is shown in Figure 22. The accuracy and solution time are comparable to the sound-soft scattering case.

10.4 Eigenvalues of a rounded ridge waveguide

The TM or TE modes and dispersion curves of a perfect electric conductor waveguide can be found from the Dirichet or Neumann eigenvalue problem

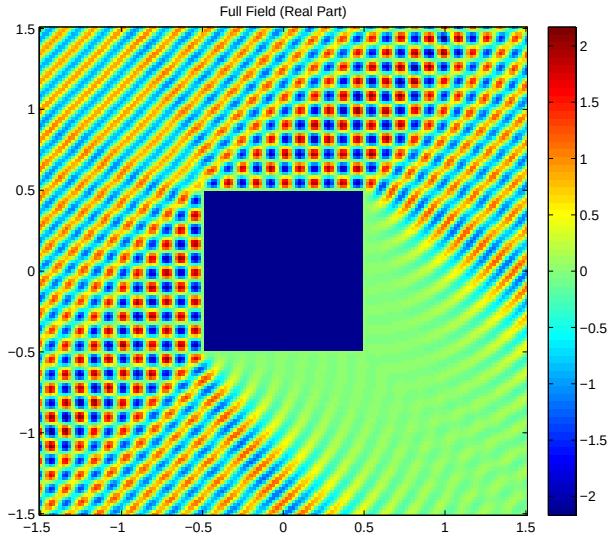

Figure 21: The solution of the scattering problem on the unit square with sound-soft boundary conditions.

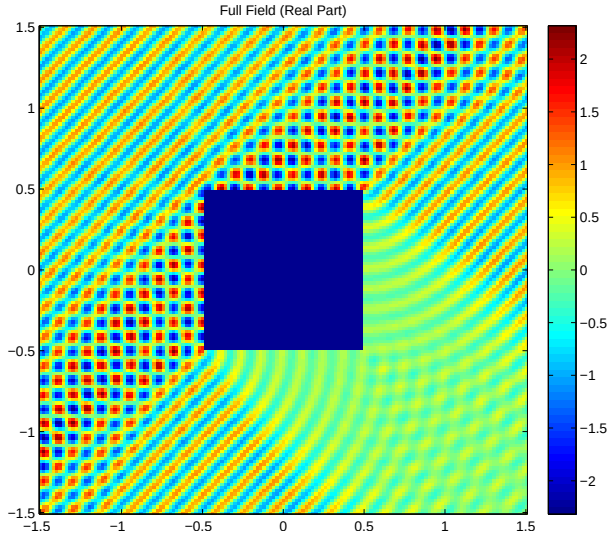

Figure 22: The solution of the scattering problem on the unit square with sound-hard boundary conditions.

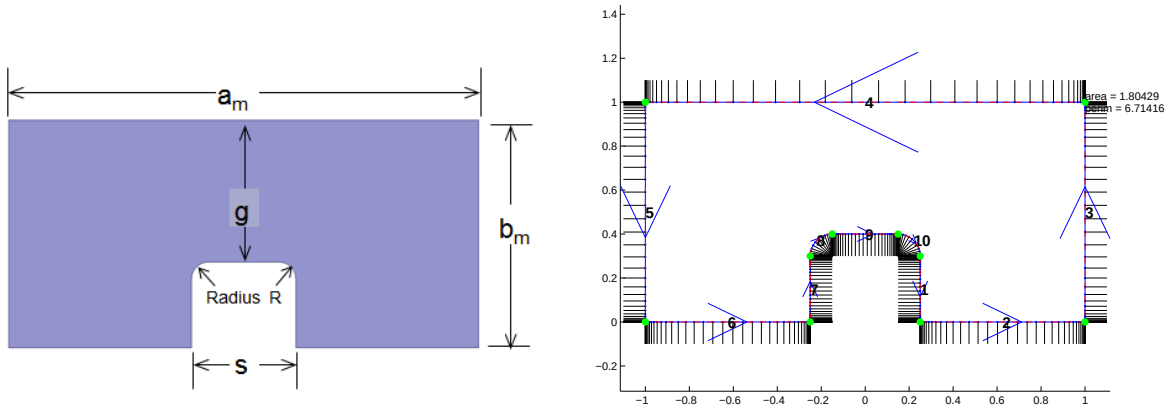

Figure 23: Geometry for the ridge guide eigenvalue problem, and the ten segments that represent it in `MPSpack`.

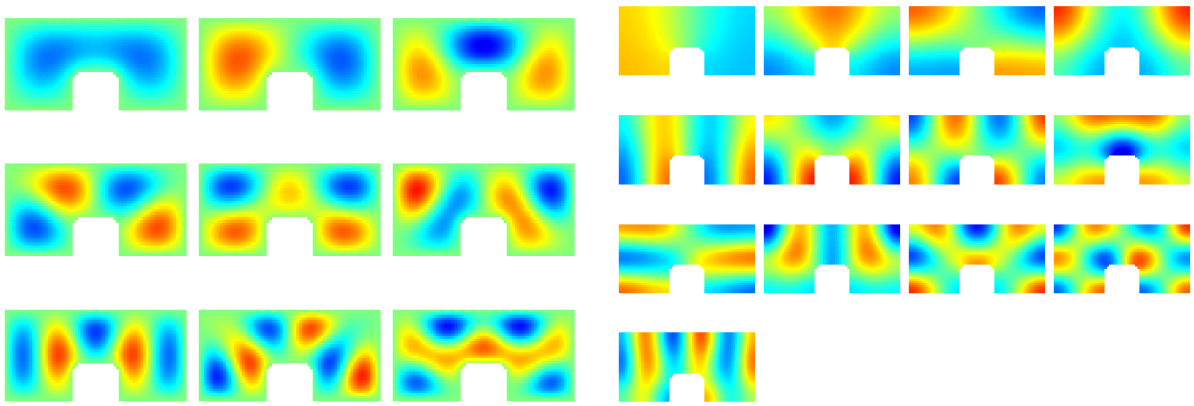

Figure 24: The lowest Dirichlet (left) and Neumann (right) eigenmodes for the ridge guide eigenvalue problem.

for the guide cross-section. Fig. 23 shows a typical example coming from engineering. The first author thanks Dr Peter S. Simon of Space Systems/Loral for this nice example problem. Since there are corners and jumps in curvature, we will need to use quadratures appropriate for layer potentials with corners.

For the solution, we set up ten segments for the boundary, using Kress' corner-adapted reparametrization (`'pc'`) of the Kress quadrature rule (make sure you have the latest version of `MPSpack` for this).

```
bc = 'D';       % boundary conditions: 'D' or 'N'
o.kressq=5;     % corner-packing parameter for Kress reparametrization

n = 48;                                     % convergence param
am = 2; bm = 1; s = 0.5; R = 0.1; g = 0.6; % ridge waveguide params
h = 1i*(bm-g-R);                            % y-shift of centers
ss = segment.polyseglist(n, [s/2+h, s/2, am/2, am/2+1i*bm, ...
  -am/2+1i*bm, -am/2, -s/2, -s/2+h], 'pc', o);
seg = [ss(1:end-1) segment(n/2,[-s/2+R+h,R,pi,pi/2],'pc',o) ...
  segment(n,[-s/2+R+h+1i*R, s/2-R+h+1i*R],'pc',o) ...
  segment(n/2,[s/2-R+h,R,pi/2,0],'pc',o)];
```

Notice that since $n/2$ points are used on the arcs, and Kress is coded only for even numbers of points, that 4 must divide n.

We then proceed as in Section 9. The code is very short.

```
d = domain(seg, 1);
seg.setbc(-1, bc);              % BC's applied on inside
p = evp(d);                     % sets up eigenvalue problem object
if bc=='D', d.addlayerpot(seg, 'd'); else, d.addlayerpot(seg, 's'); end
o.modes = 1;
o.iter = 0;  % can set to 1 if no degeneracies
o.tol = 1e-4; % make sure to capture even bad ones
p.solvespectrum([2 8], 'fd', o);
p.showmodes;
```

The resulting modes for the two boundary conditions D or N are shown in Fig. 24. The computations take around 30 seconds. The Dirichlet case gets 9 digits of accuracy, but Neumann only 4-5 digits.

10.5 Dirichlet eigenvalues of a multiply-connected domain

We refer the reader to the code `examples/gasketevp.m` which uses the Fredholm-determinant method to compute Dirichlet eigenvalues of a smooth

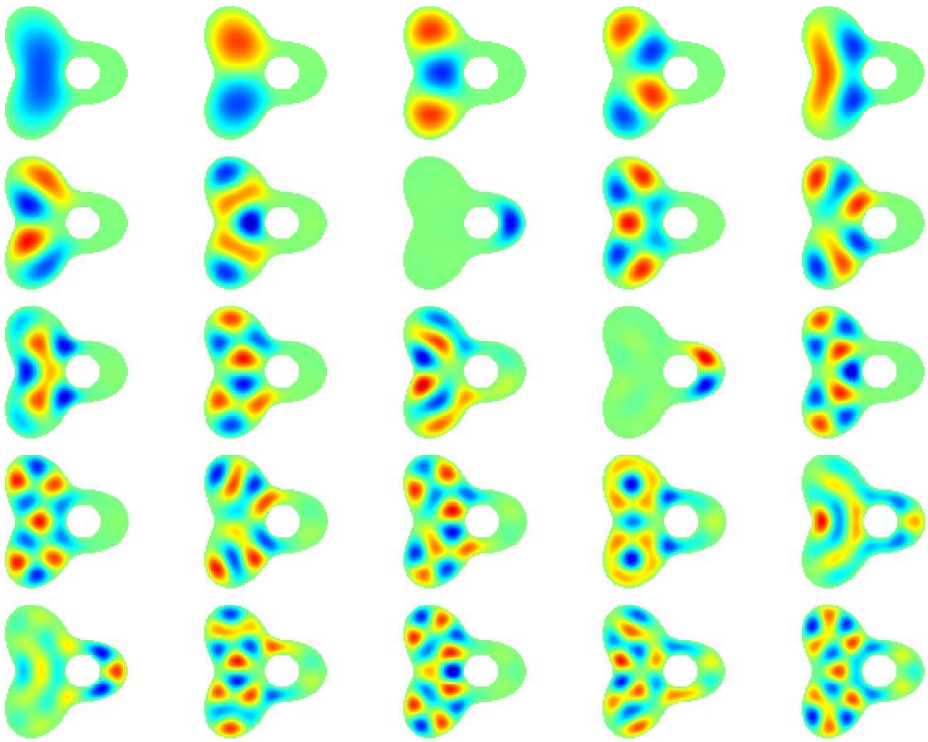

Figure 25: Lowest 25 Dirichlet eigenmodes of a smooth gasket domain, computed to 14 digit accuracy in 20 seconds.

gasket domain, with results as shown in Fig. 25. This was done by the first author during collaboration with Braxton Osting. Note that this method is not yet robust, since interior resonances of the holes are not prevented.

References

[1] A. H. Barnett and T. Betcke, *Stability and convergence of the Method of Fundamental Solutions for Helmholtz problems on analytic domains*, J. Comput. Phys., 227 (2008), pp. 7003–7026.

[2] A. H. Barnett and L. Greengard, *A new integral representation for quasi-periodic scattering problems in two dimensions*, BIT Numer. Math., 51 (2011), pp. 67–90.

[3] A. H. Barnett and A. Hassell, *Fast computation of high frequency Dirichlet eigenmodes via the spectral flow of the interior Neumann-to-Dirichlet map*, Comm. Pure Appl. Math., 67 (2014), pp. 351–407.

[4] T. Betcke, *Computations of Eigenfunctions of Planar Regions*, PhD thesis, Oxford University, UK, 2005.

[5] T. Betcke and L. N. Trefethen, *Reviving the method of particular solutions*, SIAM Rev., 47 (2005), pp. 469–491.

[6] D. Colton and R. Kress, *Inverse acoustic and electromagnetic scattering theory*, vol. 93 of Applied Mathematical Sciences, Springer-Verlag, Berlin, second ed., 1998.

[7] S. C. Eisenstat, *On the rate of convergence of the Bergman-Vekua method for the numerical solution of elliptic boundary value problems*, SIAM J. Numer. Anal., 11 (1974), pp. 654–680.

[8] J. D. Jackson, *Classical Electrodynamics*, Wiley, 3rd ed., 1998.

[9] C. M. Linton and I. Thompson, *Resonant effects in scattering by periodic arrays*, Wave Motion, 44 (2007), pp. 165–175.

[10] V. Rokhlin, *Solution of acoustic scattering problems by means of second kind integral equations*, Wave Motion, 5 (1983), pp. 257–272.