MPSpack tutorial

Alex Barnett* and Timo Betcke†

July 23, 2009

Abstract

This is a short tutorial showing how Laplace and Helmholtz boundary-value problems may be numerically solved simply and accurately with the MPSpack toolbox for MATLAB. We assume basic familiarity with MATLAB and with partial differential equations.

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 the mathematics behind the solution techniques, focusing on computing and plotting useful PDE solutions.

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. All the code examples in this document, and code to generate the figures, is found in the matlab files examples/tut_*.m

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])
```

*Department of Mathematics, Dartmouth College, Hanover, NH, 03755, USA
†Department of Mathematics, University of Reading, Berkshire, RG6 6AX, UK

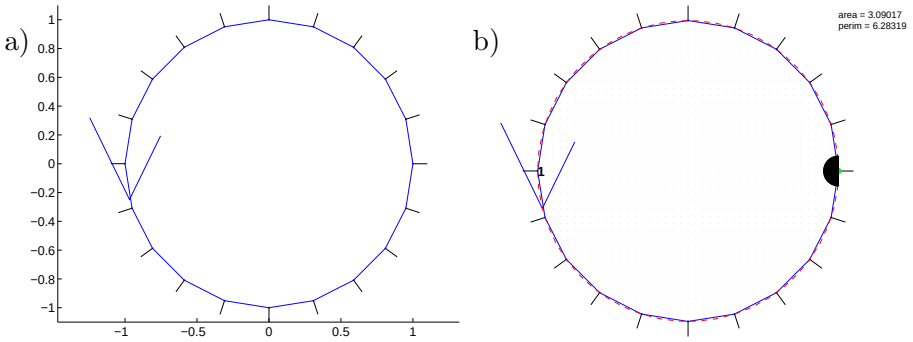


Figure 1: a) circular closed segment, b) unit disc domain. Both have a periodic trapezoidal quadrature rule with $M = 20$ quadrature points

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

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

[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);`

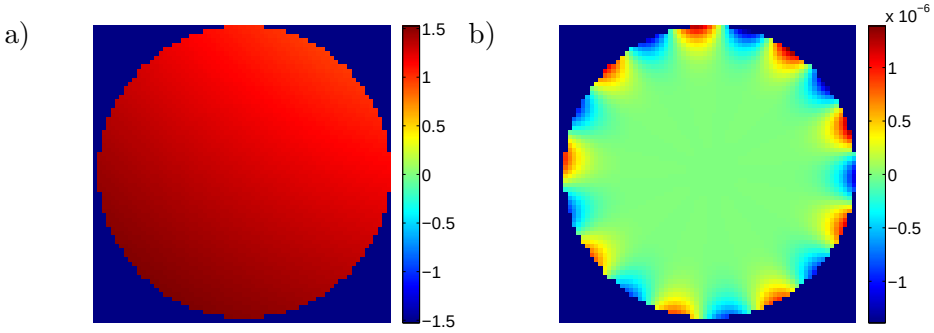


Figure 2: a) Numerical solution field u, b) pointwise error $u-f$, for Laplace's equation in the unit disc with $M = 20$ quadrature points and 8th-order harmonic polynomials.

8th-order harmonic 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.

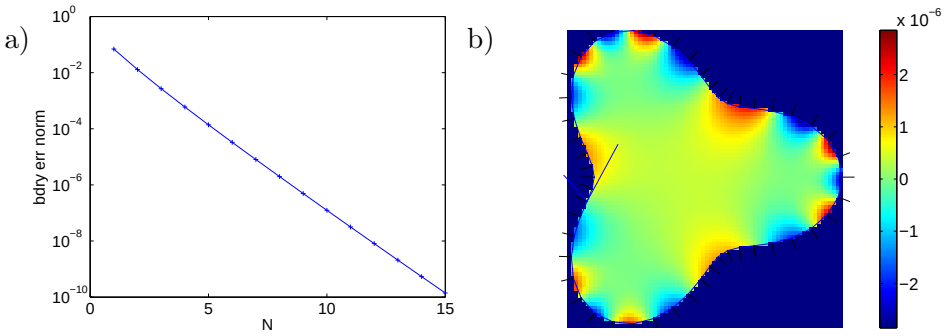


Figure 3: a) Convergence of boundary error L^2 norm for harmonic polynomials for Laplace equation in the unit disc, b) solution error for same boundary data f in a smooth star-shaped 'trefoil' domain (normals also shown).

```
p.showsolution;
```

The software chose an appropriate grid covering the domain (points outside the domain are made transparent), 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 and figure,

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

As Fig. 3a shows, the convergence is exponential.[4]

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 by

```
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');
```

[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 [2], 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 since problem setup time is very rapid.

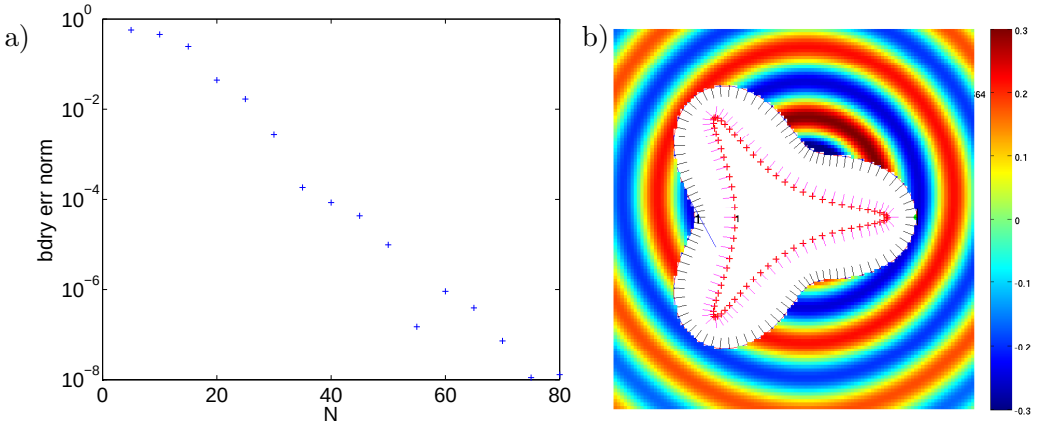


Figure 4: a) Convergence of exterior Helmholtz BVP with MFS basis, b) Domain boundary and MFS charge curve geometry, and (real part of) the solution field outside the domain.

Note the nested anonymous functions for mathematical clarity. Note 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: a exterior Helmholtz BVP with Neumann boundary data, and the Sommerfeld radiation condition [3]. 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 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));  % a radiative Helm soln
tref.setbc(1, 'D', [], @(t) f(tref.Z(t)));            % exterior Dirichlet data
```

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

$$\Phi(\mathbf{x}) = \frac{i}{4} H_0^{(1)}(k|\mathbf{x}|) \tag{1}$$

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}.

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',

[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. See [1].

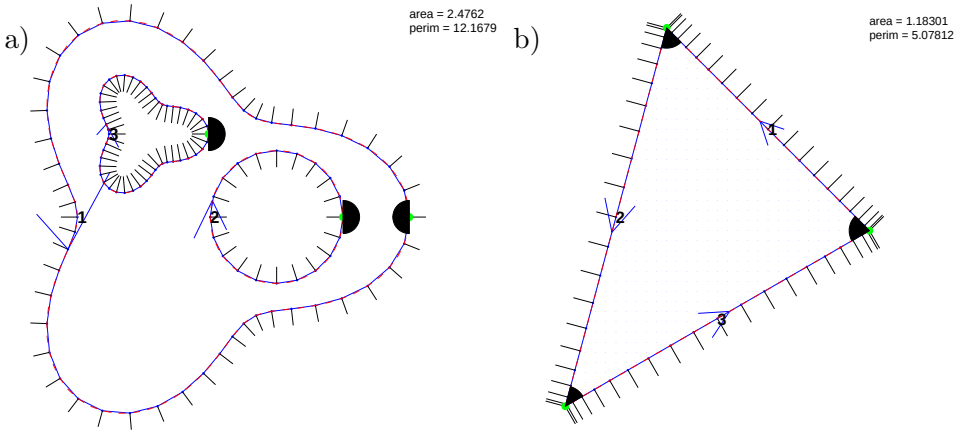


Figure 5: a) A multiply-connected domain. b) A polygonal domain.

```
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 copy of tref
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 Sec. 8.

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 from sending a list of three line segments to the domain command; this list may conveniently be made with a polygon constructor,

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

Since `s` is a 1-by-3 array of (handles to) segments, the 2nd argument 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 1i]);
```

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 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, since each was or to create the domain exterior to two nearby triangles,

```
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

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

[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.

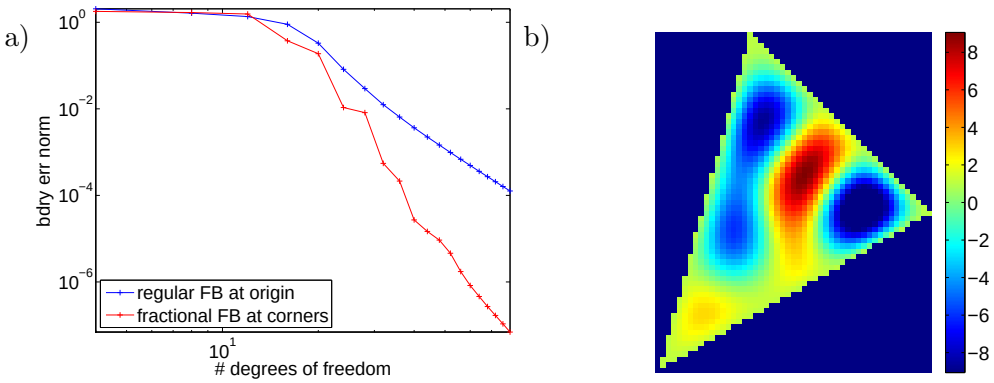


Figure 6: a) Comparing convergence of boundary error norm for Fourier Bessel vs corner-adapted fractional-order Fourier Bessel basis sets in a triangle with unity Dirichlet boundary data, for Helmholtz BVP with $k = 10$, on log-log axes. b) The solution function.

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) 1);              % 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)
  tri.bas{1}.N = Ns(i); p.solvecoeffs; r(i) = p.bcresidualnorm;
end
figure; loglog(2*Ns, r, '+-'); xlabel('# dofs'); ylabel('bdry err norm');
```

This produces the algebraic convergence[9] shown in blue in Fig. 6a.

We now show how corner-adapted basis sets may enhance the above basis set to achieve superior convergence and hence more efficiency. We clear the previous basis set from the domain and add fractional-order (i.e. 'wedge' expansion) Fourier Bessel bases at two of the corners, of both cos

[9]The power law for convergence is related to the largest corner angles and is discussed in [4].

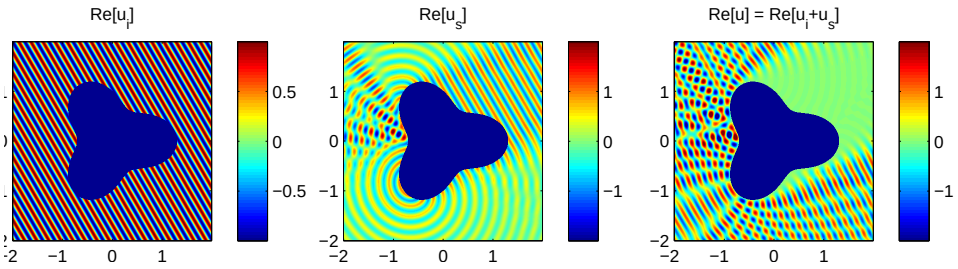


Figure 7: Sound-soft (homogeneous Dirichlet BC) scattering from a smooth trefoil at $k = 30$ using an MFS basis. Left: incident wave u_{inc}. Center: scattered wave u_s. Right: their sum, the total solution field u.

and sin type.[10] Repeating the convergence study and comparing against the previous data is easy,

```
tri.clearbases; opts.rescale_rad = 2.0; opts.cornermultipliers = [1 1 0];
tri.addcornerbases([], opts);
Ns = 1:20; 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 data is shown in red in Fig. 6a, and is much faster. [initially it is superalgebraic. TIMO: WHY IS IT NOT SUPER-ALGEBRAIC ALL THE WAY DOWN? IT IS ALSO too SENSITIVE to rescalerad - WHY?] 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 the 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)

[10] Since only one of the corners is singular, it is possible to omit a corner expansion from one of the other corners, for instance the 3rd one. This done with the options `opts.cornermultipliers = [1 1 0];`.

Helmholtz solution u_{inc}. The unknown u_s is then the solution to the radiative exterior BVP solved in Sec. 4 (see Sec. 8.2.1 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,

```
opts.tau = 0.05; d.addmfsbasis(tref, 200, opts);  % basis set for ext domain
s.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,

```
s.setbc(1, 'N', []);                    % homogeneous Neumann: sound-hard
s.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 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);
```

[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);`

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

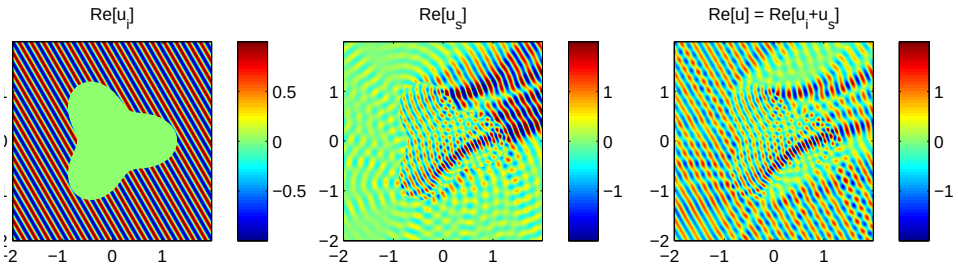


Figure 8: Transmission scattering from a smooth trefoil with refractive index (interior to exterior wavenumber ratio) 1.5 and exterior wavenumber $k = 30$, using MFS bases inside and out. Left: incident wave u_{inc}. Center: scattered wave u_s. Right: total solution field u.

```
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. This is also appropriate for acoustics when u represents the pressure field. 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 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. 8.2. This enables dielectric-coated metals, dielectrics with interior holes, etc, to be solved; see Sec. 8.1.

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

7 Layer potentials

Layer potentials are representations of Helmholtz solutions involving an integral over a surface, i.e. a boundary segment [3]. They are similar 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 in a domain a single-layer potential (SLP) density

$$u(\mathbf{x}) = \mathcal{S}\sigma := \int_\Gamma \Phi(\mathbf{x}-\mathbf{y})\sigma(\mathbf{y})ds_\mathbf{y} \tag{2}$$

where ds is arclength, or double-layer potential (DLP) density

$$u(\mathbf{x}) = \mathcal{D}\tau := \int_\Gamma \frac{\partial\Phi(\mathbf{x}-\mathbf{y})}{\partial n_\mathbf{y}}\tau(\mathbf{y})ds_\mathbf{y} \tag{3}$$

where Γ is a segment, and σ and τ are functions on Γ. For example, using as Γ the trefoil segment `tref` defined in Sec. 4, a DLP is set up by

```
d = domain([], [], tref, -1); d.k = 10;    % external domain, wavenumber k
d.addlayerpot([], 'd');                    % adds DLP to bdry of d
```

The coefficients in `p.co` represent density function values at the quadrature points.

TO FINISH.

8 Examples

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

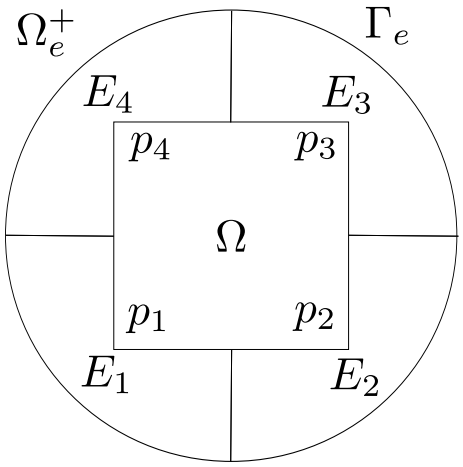
8.1 Transmission scattering with metallic inclusions and air holes

DO THE LAST CASE IN test/testscattering.m

NOT HARD.

8.2 Acoustic scattering from the unit square

This example introduces a new feature: the use of decomposition of a region with constant wavenumber into subdomains connected by fictitious boundaries.



8.2.1 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.

$$\begin{aligned} \Delta u + k^2 u &= 0 \quad \text{in } \mathbb{R}^2 \backslash \Omega & (4) \\ u &= 0 \quad \text{on } \partial\Omega & (5) \\ \frac{\partial u_s}{\partial r} - iku_s &= o(r^{-1/2}), & (6) \end{aligned}$$

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.

To achieve high accuracy we cannot simply use fundamental soluitions 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 8.2.1. The idea is that each of the elements E_i only contains one corner 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.

8.2.2 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 8.2.1.

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

```
k = 50;      % Wavenumber
r = 1.0;     % Radius of outer circle
```

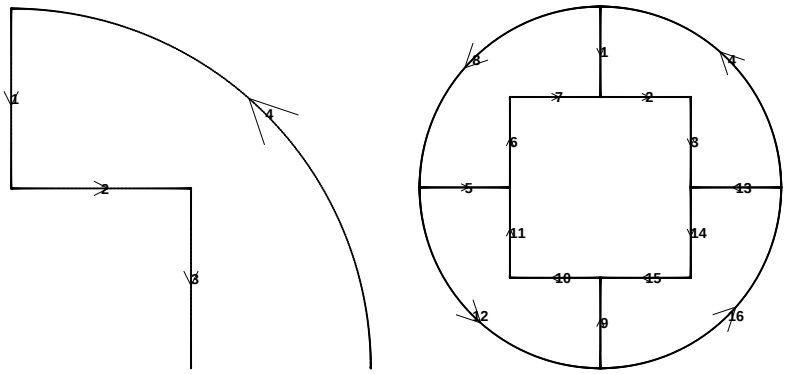


Figure 9: The segments in the right plot are created by rotating the segments shown in the left plot.

```
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.

First we define the segments of one element (E_3 in Figure 8.2.1). This is done with the following three commands.

```
s = segment.polyseglist(M, [1i*r 1i*a a+1i*a a r], 'g');
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 shown in the left plot of Figure 9. We now rotate this element three times to obtain the segments showin in the right plot of Figure 9.

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.

Setting up the compatibility conditions is now 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.[14]

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);
```

[14] 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.

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.

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.

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

```
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.

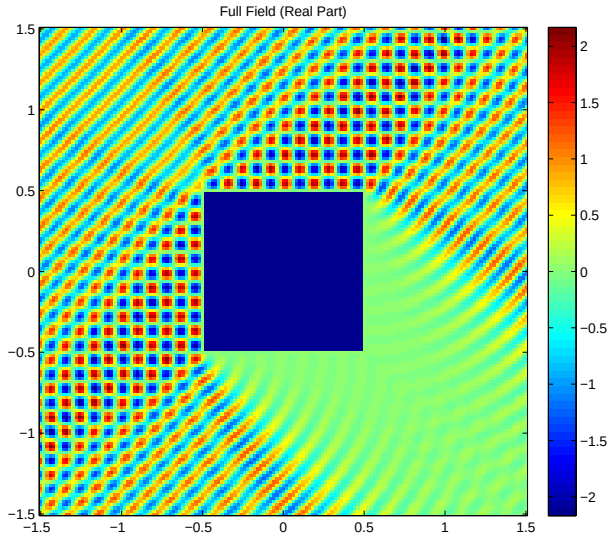


Figure 10: The solution of the scattering problem on the unit square with sound-soft boundary conditions.

The output of the example problem is shown in Figure 10. 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 11. The accuracy and solution time are comparable to the sound-soft scattering place.

8.3 Transmission scattering from multiple dielectric discs

TIMO TO DO ?

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.

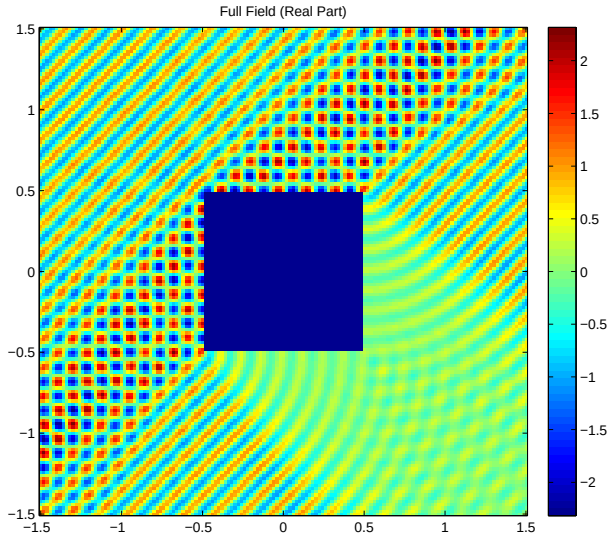


Figure 11: The solution of the scattering problem on the unit square with sound-hard boundary conditions.

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

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

[4] 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.