Improved Efficiency and Accuracy of the Magnetic Polarizability Tensor Spectral Signature Object Characterisation for Metal Detection

J. Elgy[a,*], P. D. Ledger[a]

[a]*School of Computer Science and Mathematics, Keele University, Keele, Staffordshire, UK*

Abstract

Keywords:

1. Introduction

1.1. Model Problem

$$\begin{aligned} \nabla \times \nabla \times \boldsymbol{E} - k^2 \boldsymbol{E} &= \boldsymbol{0} && \text{in } \Omega \in \mathbb{R}^3 \\ \boldsymbol{n} \times \boldsymbol{E} &= \boldsymbol{n} \times \boldsymbol{E}^{(exact)} && \text{on } \Gamma = \partial\Omega \end{aligned} \tag{1}$$

with $\boldsymbol{E}^{(exact)} = \boldsymbol{p}e^{i\boldsymbol{k}\cdot\boldsymbol{x}}$ being the known solution for the electric field at position $\boldsymbol{x}$ for wavevector $\boldsymbol{k} = [k_x, k_y, k_z]$ and perpendicular amplitude $\boldsymbol{p} = [p_x, p_y, p_z]$. For the standard $\boldsymbol{H}(\text{curl})$ conforming finite element space [2] the weak form of this problem is: Find $\boldsymbol{u} \in \boldsymbol{H}_D(\text{curl})$ such that

$$\int_\Omega \nabla \times \boldsymbol{u} \nabla \times \boldsymbol{v} \, \mathrm{d}\Omega - \int_\Omega k^2 \boldsymbol{u}\boldsymbol{v} \, \mathrm{d}\Omega = \boldsymbol{0}, \tag{2}$$

for all $\boldsymbol{v} \in \boldsymbol{H}_0(\text{curl})$ and $k^2 = |\boldsymbol{k}|^2 > 0$ being homogeneous and isotropic in Ω. Here, $\boldsymbol{H}(\text{curl})$ is the high order space defined by

$$\boldsymbol{H}(\text{curl}) := \left\{ \boldsymbol{a} \in \mathrm{L}^2(\Omega)^3 | \nabla \times \boldsymbol{a} \in \mathrm{L}^2(\Omega)^3 \right\},$$

where $L^2(\Omega)$ denotes the space of square integrable functions. Considering Dirichlet boundary conditions and setting $\boldsymbol{v}$ to vanish on the boundary, the appropriate subspaces for this problem are

$$\begin{aligned} \boldsymbol{H}_D(\text{curl}) &:= \left\{ \boldsymbol{a} \in \boldsymbol{H}(\text{curl}) | \boldsymbol{n} \times \boldsymbol{a} = \boldsymbol{n} \times \boldsymbol{E}^{(\text{exact})} \text{ on } \Gamma \right\} \\ \boldsymbol{H}_0(\text{curl}) &:= \left\{ \boldsymbol{a} \in \boldsymbol{H}(\text{curl}) | \boldsymbol{n} \times \boldsymbol{a} = \boldsymbol{0} \text{ on } \Gamma \right\}. \end{aligned}$$

The Galerkin finite element discretisation of the variational statement (2) is the large linear system

$$\mathbf{A}\mathbf{q} = \mathbf{r} \tag{3}$$

where $\mathbf{A}$ is a large sparse matrix of size N_d. The symmetric sparse matrix $\mathbf{A}$ is indefinite, and consequently may be difficult to solve.

Should expand

2. Software

The computational resources were used to perform the simulations in this paper correspond to workstations with the following specifications: Intel i7-9700K CPU with a clock speed of 3.60GHz with 64GB DDR4 RAM.

The software used for this work `NGSolve` (version 6.2.2302) and `Netgen` (version 6.2.2302) [4, 6, 3], `SciPy` [5] (version 1.10.1), `NumPy` (version 1.24.2) [1].

** *Email addresses:* `j.elgy@keele.ac.uk` (J. Elgy), `p.d.ledger@keele.ac.uk` (P. D. Ledger)

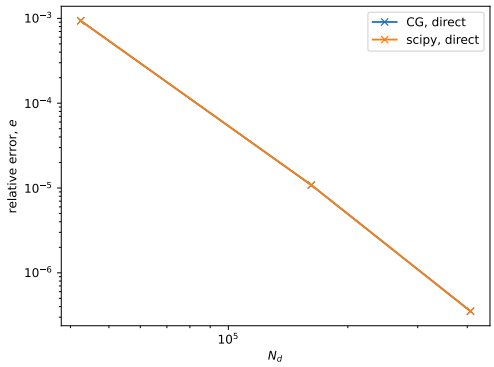

Figure 1: Non-mangetic non-conducting sphere of unit radius discretised using 14989 unstructured tetrahedral elements using $p = 0, 1, 2, 3$. A comparison between GMRES and conjugate gradient solvers, where differences are indistinguishable on this scale.

3. Conversion to Scipy Iterative Solvers

The iterative solvers available in `NGSolve`, notably `CGSolver` and `GMRESSolver` do not allow for the execution of arbitrary callback functions in the same way as many other solvers from other established python libraries. For this reason, it is difficult to retain the residual at each iteration of the `NGSolve` solvers. Furthermore, the `GMRESSolver` does not appear to have a restart option available to the API and is in general poorly documented. For this reason, the solving of the linear system used in static condensation is instead implemented using and

The NGSolve implementation of GMRES also gives noticeably different results to the CGSolver, even when using the same direct preconditioner, so I don't really trust it.

3.1. Choice of Preconditioner

In this report, we consider 4 different preconditioners, as implemented by `NGSolve`. A direct inverse preconditioner, where $\mathbf{P} = \mathbf{A}$ and $\mathbf{P}^{-1}\mathbf{A} = \mathbb{I}$, a Jacobi preconditioner, where $\mathbf{P} = \text{diag}(\mathbf{A})$, a Multigrid preconditioner, and the Balancing Domain Decomposition with Constraints (BDDC) preconditioner. References

4. Numerical Results

In this section, we consider a range of numerical examples to illustrate the performance of the different preconditioners.

4.1. Non-Magnetic Sphere

We first consider the case of a sphere of radius 1 centred at $\boldsymbol{x} = \mathbf{0}$, discretised using $h = 0.04$, resulting in Ω being discretised by 14989 unstructured tetrahedral elements. The amplitude and wavevector are chosen as $\boldsymbol{p} = [0, 1, 0]$ and $\boldsymbol{k} = [1, 0, 0]$, respectively. We therefore expect the exact solution to have unit magnitude and a wavelength of 2π. Using $TOL = 1 \times 10^{-12}$ and a direct inverse preconditioner we compare the performance of the `NGSolve` conjugate gradient solver (which is valid given the applied preconditioner) with the `SciPy` GMRES solver. Figure 1 shows the error between the approximate and exact solutions in the L_2 norm,

$$e = \frac{\left(\int_\Omega \left(\boldsymbol{E}^{(hp)} - \boldsymbol{E}^{(exact)}\right) \cdot \overline{\left(\boldsymbol{E}^{(hp)} - \boldsymbol{E}^{(exact)}\right)}\, \mathrm{d}\Omega\right)^{1/2}}{\left(\int_\Omega \left(\boldsymbol{E}^{(exact)}\right) \cdot \overline{\left(\boldsymbol{E}^{(exact)}\right)}\, \mathrm{d}\Omega\right)^{1/2}}, \tag{4}$$

for the aforementioned discretisation and $p = 0, 1, 2, 3$ for both the GMRES and conjugate gradient solvers. The figure shows rapid decay of the error as p is increased. Similar agreement between the two solvers is also observed for other configurations. Due to the application of the direct inverse preconditioner, the solvers converge in 1 or 2 iterations.

Having established that the `SciPy` GMRES solver and the `NGSolve` CG Solver are in agreement, we now consider the case of the BDDC preconditioner, when used in conjunction with the `SciPy` GMRES solver.

First, in Figure 2, we establish that the error, e, is invariant to the choice of preconditioner, for a sufficiently accurate solution to (3). The figure shows that applying both the BDDC and Multigrid preconditioners results in an accurate approximation to (2), whereas the Jacobi (local) preconditioner does not. This can be explained by the solution to (3) not being sufficiently accurate. A demonstration of the convergence behaviour for this problem can be seen in Figure 3, where the relative L_2 residual is plotted for each iteration for each preconditioner for $p = 0, 1, 2, 3$. The figure shows that while the BDDC and multigrid preconditioners converge quickly, the Jacobi preconditioner does not, and is unsuitable for this problem. In the case of the Multigrid and BDDC preconditioners, the lowest order terms are solved directly, as a result the $p = 0$ case is solved in 1 iteration.

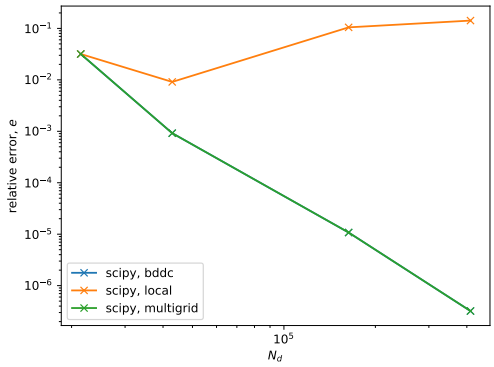

Figure 2: Non-mangetic non-conducting sphere of unit radius discretised using 14989 unstructured tetrahedral elements using $p = 0, 1, 2, 3$. Computed error estimate, e for BDDC, Jacobi (local), and Multigrid preconditioners.

From the figure, we see that the Multigrid preconditioner results in the fewest number of iterations, however this does not account for the computation time required for each iteration. Figure 4 shows the wall clock computation time for each iteration of the GMRES solver for this sphere problem using (a) $p = 0$, (b) $p = 1$, (c) $p = 2$, and (d) $p = 3$. Note that this does not include the time required to construct the preconditioner. Timings were performed using a single threaded implementation of the iterative solver using the workstation and software provided in Section 2. Similar to Figure 3, we observe that the Multigrid preconditioner is (slightly) faster than both the BDDC and Jacobi preconditioners.

This may be a consequence of the wavenumber, and by extension wavelength, so we next consider the case where $|\boldsymbol{k}|$ is increased from 1 to 2 to 4. In these tests, the discretisation of the sphere is kept constant (using radius 1 and 15243 unstructured tetrahedra); the increase in wavenumber is reported in terms of the number of elements per wavelength. Figure 5 shows a comparison for the same sphere discretisation, with $h = 0.08$, where the wavelength is increased from 2π to 4π to 8π between the Multigrid and BDDC preconditioners with respect to the number of iterations required to reach a stopping tolerance of 1×10^{-12}

Appendix A. Code for NGSolve and SciPy Solvers

The following two code snippets are equivalent.

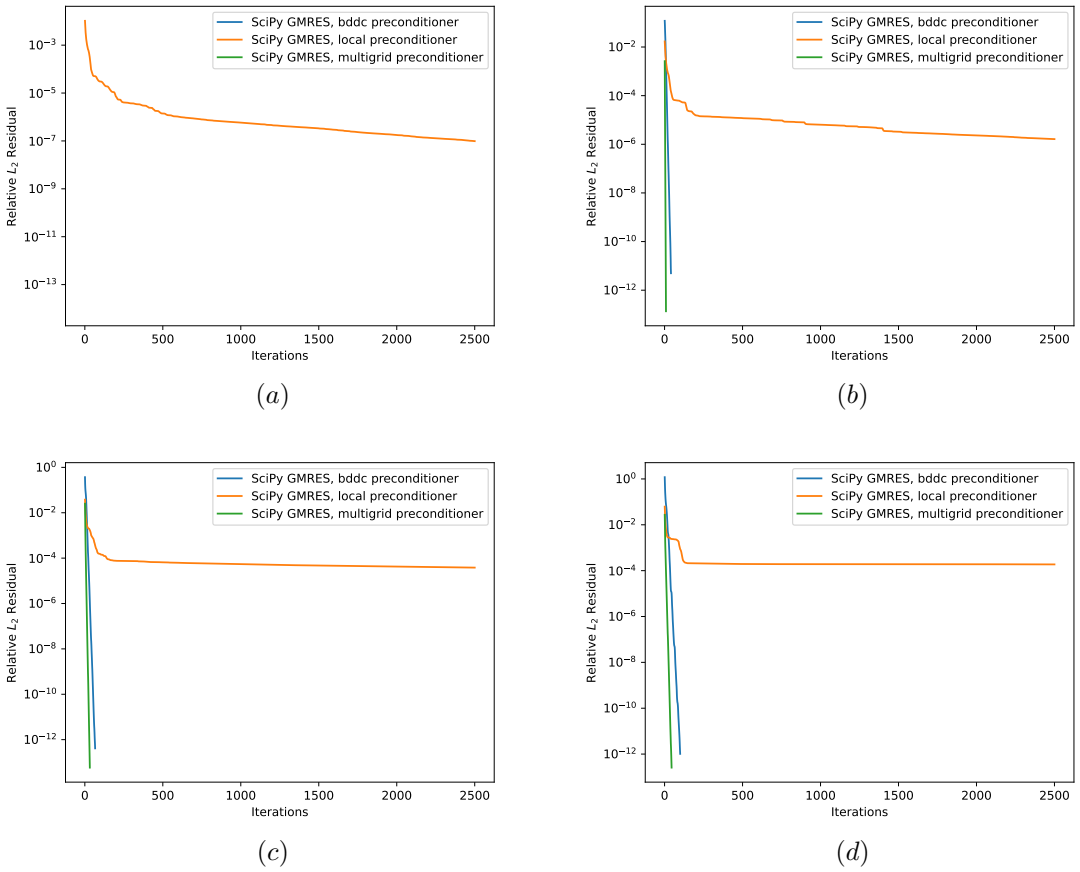

Figure 3: Non-mangetic non-conducting sphere of unit radius discretised using 15243 unstructured tetrahedral elements using $p = 0, 1, 2, 3$. Computed residual to (3) for BDDC, Jacobi (local), and Multigrid preconditioners. Figure shows that both BDDC and Multigrid converge rapidly, and that the local preconditioner does not result in convergence. (a) $p = 0$, (b) $p = 1$, (c) $p = 2$, (d) $p = 3$,

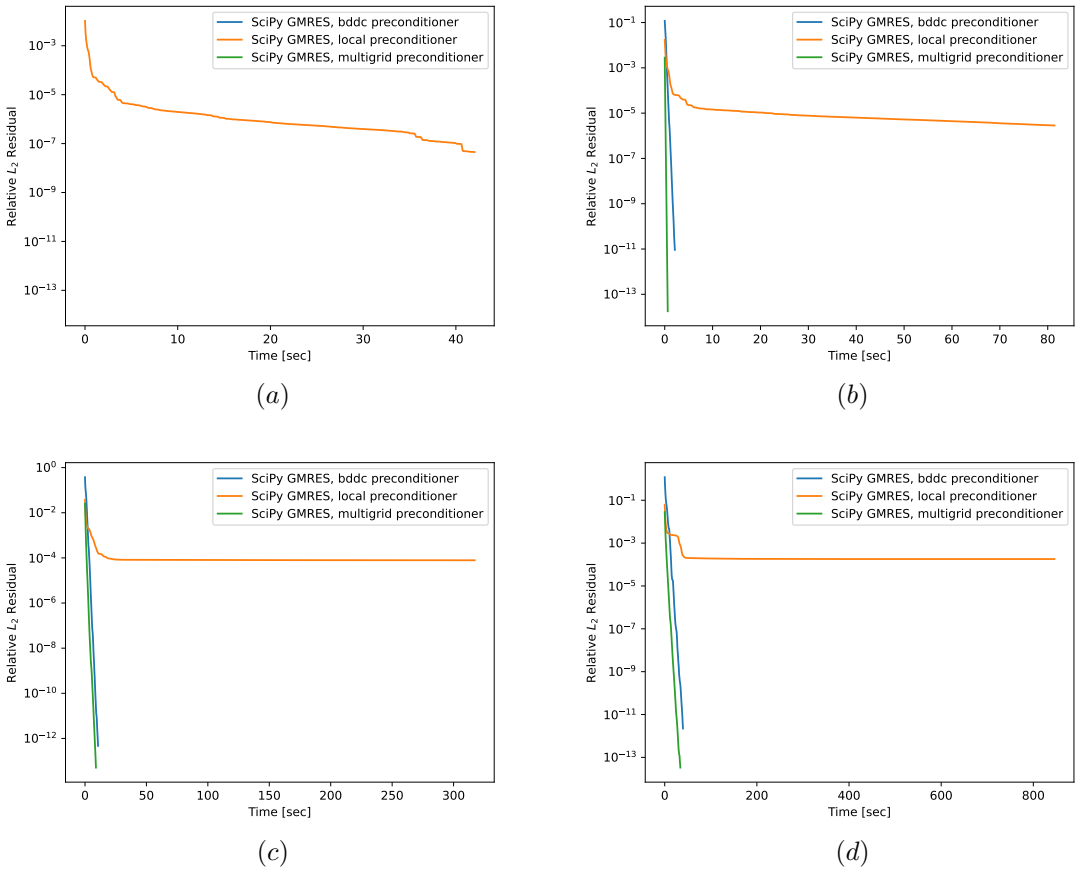

Figure 4: Non-mangetic non-conducting sphere of unit radius discretised using 15243 unstructured tetrahedral elements using $p = 0, 1, 2, 3$. Computation time for BDDC, Jacobi (local), and Multigrid preconditioners. Figure shows that both BDDC and Multigrid converge rapidly, and that the local preconditioner does not result in convergence. (a) $p = 0$, (b) $p = 1$, (c) $p = 2$, (d) $p = 3$,

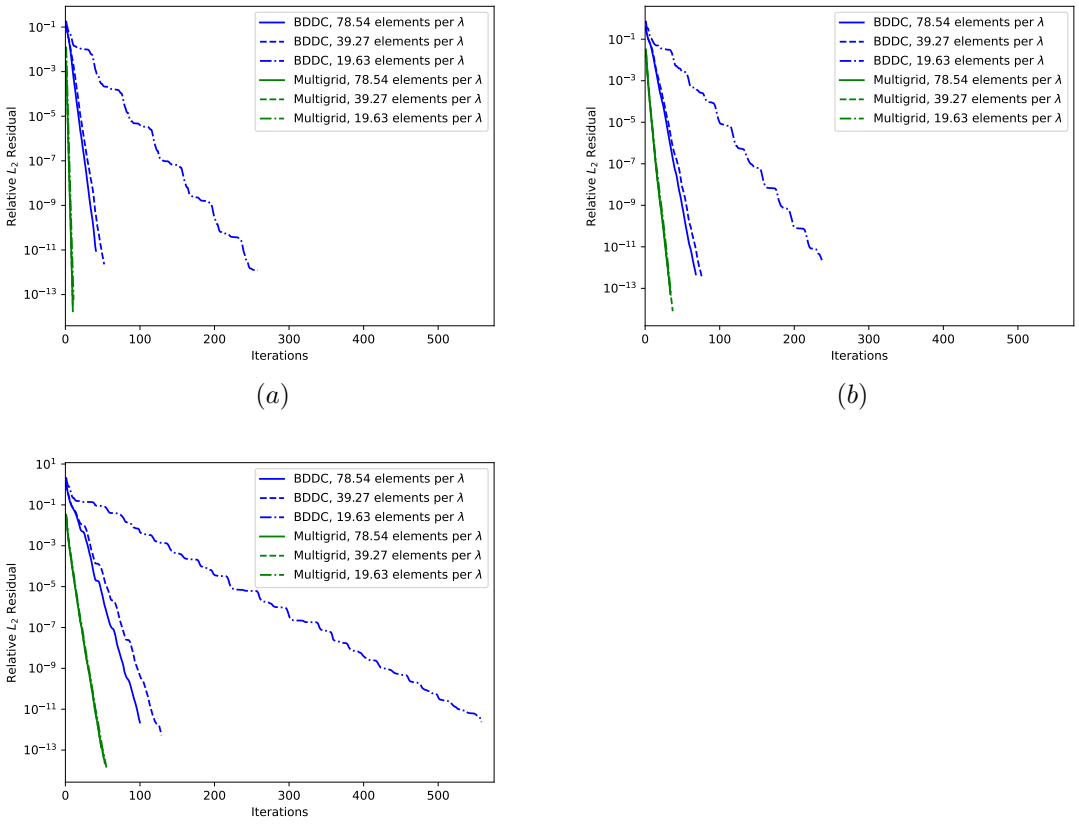

Figure 5: Non-mangetic non-conducting sphere of unit radius discretised using 15243 unstructured tetrahedral elements using $p = 1, 2, 3$. Number of iterations for BDDC and Multigrid preconditioners when considering $|\boldsymbol{k}| = 1, 2$, and 4. Figure shows that both BDDC and Multigrid converge rapidly, with the number of iterations required for the BDDC preconditioner reducing as the number of elements per λ increases. (a) $p = 1$, (b) $p = 2$, (c) $p = 3$.

```
# For assembled linear form f and bilinear form a for solution vector u.
r = f.vec.CreateVector()
r.data = f.vec - a.mat * u.vec
r.data += a.harmonic_extension_trans * r

# CGSolver(.) * r is equivalent to solving preconditioned Ax=r
u.vec.data += CGSolver(A.mat, P.mat, precision=1e-10) * r
u.vec.data += a.harmonic_extension * u.vec
u.vec.data += a.inner_solve * r
```

Listing 1: Static condensation using NGSolve

```
# For assembled linear form f and bilinear form a for solution vector u.
r = f.vec.CreateVector()
r.data = f.vec - a.mat * u.vec
r.data += a.harmonic_extension_trans * r

# The projector is used to remove non-local and non-Dirichlet degrees of freedom that
    should not participate in the solve
q = r.CreateVector()
pre = Projector(mask=fes.FreeDofs(coupling=True), range=True)

tmp1 = F.vec.CreateVector()
tmp2 = F.vec.CreateVector()

# A linear operator that returns Av for the sparse matrix A
def matvec(v):
  tmp1.FV().NumPy()[:] = v
  tmp2.data = A.mat * tmp1
  tmp2.data = pre * tmp2
  return tmp2.FV().NumPy()

r.data = pre * res
A_linop = sp.sparse.linalg.LinearOperator((A.mat.height, A.mat.width), matvec)

# Solve Ax = r
q.FV().NumPy()[:], OutputStatus = sp.sparse.linalg.gmres(A_linop, r.FV().NumPy(), tol=1e
    -10, M=P.mat)
u.vec.data += q

u.vec.data += a.harmonic_extension * u.vec
u.vec.data += a.inner_solve * r
```

Listing 2: Static condensation using SciPy

References

[1] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[2] Monk, P., and Zhang, Y. Finite Element Methods for Maxwell's Equations. *Encyclopedia of Computational Mechanics Second Edition* (2017), 1–20.

[3] Schöberl, J. NETGEN - an advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science 1(1)* (1997), 41–52.

[4] Schöberl, J. C++11 implementation of finite elements in NGSolve. Tech. rep., ASC Report 30/2014, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2014.

[5] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors.

SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272.

[6] Zaglmayr, S. *High Order Finite Elements for Electromagnetic Field Computation.* PhD thesis, Johannes Kepler University Linz, 2006.