UK Atomic Energy Authority

ExCALIBUR

Identification of suitable preconditioner techniques

M2.7.2

The report describes work for ExCALIBUR project NEPTUNE at Milestone 2.7.2. This binds the following reports 2047353-TN-01[1], 2047353-TN-02[2], 2047353-TN-03[3] and 2047353-TN-04[4] as of August 27, 2021.

In scientific computing the discrete solution of a system of Partial Differential Equations (PDEs) is often given by the solution $\vec{x}$ of the equation

$$A\vec{x} = \vec{b} \tag{1}$$

where the matrix A is the representation of the differential operators in the chosen discretisation and $\vec{b}$ is a known vector for the given problem. For very small system sizes, where the size of A is small, the system of equations described by Equation (1) can be solved using direct methods which explicitly form A^{-1}, typically using Gaussian elimination where the cost scales as $\mathcal{O}(\text{Size})^3$. However for larger system sizes direct methods become computationally infeasible and iterative methods, that form a sequence of increasingly better solutions $\{\vec{x}_i\}_{i=1}^{N}$, are employed as an alternative approach.

As discussed in this report, the number of iterations N required to reach a solution $\vec{x}_N$ such that $|A\vec{x}_N - \vec{b}| < \tau$, for an appropriate norm and tolerance τ is dependent on the matrix A. The aim of preconditioning is to apply left and/or right matrices, or equivalently operators, P_L and P_R to form the system

$$P_L A P_R \vec{y} = P_L \vec{b}, \text{ where } \vec{x} = P_R \vec{y} \tag{2}$$

the solution of which is also a solution to the system described in Equation (1). An iterative solver applied to Equation (2) produces the sequence of iterations $\{\vec{x'}_i\}_{i=1}^{N_P}$ where $|A\vec{x'}_{N_P} - \vec{b}| < \tau$ as required. The aim is to find matrices P_L and/or P_R such that the computation of the sequence $\{\vec{x'}_i\}_{i=1}^{N_P}$ is computationally cheaper than the computation of the original sequence $\{\vec{x}_i\}_{i=1}^{N}$, typically because $N_p \ll N$. Typically the construction and application of the matrices P_L and or P_R is significant and the performance improvement is realised with a reduction in the number of iterations. Ideally $N_p \ll N$. As demonstrated by the results in this report, successful preconditioning techniques can vastly reduce the computational cost of solving Equation (1).

A preconditioner is regarded as successful if it decreases the computational cost of computing a desired solution. In some cases the cost reduction is very significant and the preconditioner enables the computation of a simulation which otherwise would be computationally prohibitive. In all cases, where a successful preconditioner is applied, the overall computational efficiency of the simulation is improved. Hence successful preconditioning techniques are of importance

to the NEPTUNE project as more efficient use of computational resource directly improves the simulation and modelling capability of the project.

Unfortunately the design and construction of a preconditioner for a given system is highly non-obvious and needs to be considered on a per-problem basis. This report presents an overview of different existing classes of preconditioners and gives insight into the trade-off between the computational cost of assembling and applying a preconditioner and the potential effectiveness. Some of these preconditioners cannot be applied in a matrix-free manner as they require the matrix A to be explicitly constructed. However, other preconditioners are operator-based and only require access to elements of the matrix or the ability to compute a matrix-vector product, and so are amenable to a matrix-free implementation.

This report investigates elliptic and hyperbolic problems using the BOUT++ software [5] which uses finite differences as a spatial discretisation and Nektar++ [6] which applies a spectral/hp finite element method. The report investigates the Markov Chain Monte Carlo Matrix Inversion (MCMCMI) preconditioner, a sparse approximate inverse approximation approach, applied to a non-linear diffusion test case in BOUT++. Using Nektar++ the MCMCMI preconditioner is also applied to Advection-Diffusion-Reaction and Helmholtz systems. The MCMCMI preconditioner is used in conjunction with the Generalized Minimal Residual Method (GMRES) and Biconjugate Gradient Stabilized Method (BiCGSTAB) iterative solvers. This section also investigates the application of a selection of operator-based preconditioners to a 1D continuum model of plasma flow implemented in BOUT++ under the name SD1D [7].

In the final section this report investigates the application of implicit-factorisation constraint preconditioners to non-symmetric problems. The report first extends existing theorems from the symmetric case to cover the non-symmetric case, and gives an example of how an implicit-factorisation constraint preconditioner is constructed. As a numerical test a selection of proposed preconditioners are applied to the Hasegawa-Wakatani problem in combination with a GMRES linear solver. In these numerical experiments the constraint preconditioners demonstrate a several orders of magnitude reduction in the number of linear solve iterations in comparison to a block diagonal preconditioner.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

References

[1] S. Thorne. Priority Equations and Test Cases. Technical Report 2047353-TN-01, UKAEA Project Neptune, 2021.

[2] V. Alexandrov, A. Lebedev, E. Sahin, and S. Thorne. Linear systems of equations and preconditioners relating to the NEPTUNE Programme: A brief overview. Technical Report 2047353-TN-02, UKAEA Project Neptune, 2021.

[3] M. Abalenkovs, V. Alexandrov, A. Lebedev, E. Sahin, and S. Thorne. Implicit-factorization preconditioners for NEPTUNE Programme. Technical Report 2047353-TN-03, UKAEA Project Neptune, 2021.

[4] M. Abalenkovs, V. Alexandrov, A. Lebedev, E. Sahin, and S. Thorne. Implicit-factorization preconditioners for non-symmetric problems. Technical Report 2047353-TN-04, UKAEA Project Neptune, 2021.

[5] B.D. Dudson, P.A. Hill, D. Dickinson, J.T. Parker, A. Allen, G. Breyiannia, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Liten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, D. Schwrer, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, A. Bokshi, H. Muhammed, M. Estarellas, and F. Riva. Bout++ v4.3.1. `https://doi.org/10.5281/zenodo.3727089`, March 2020.

[6] D. Moxey et al. Nektar++ website. `https://www.nektar.info`, 2020. Accessed: June 2020.

[7] One-dimensional plasma-neutral simulation for SOL and divertor studies. `https://github.com/boutproject/SD1D`, 2021. Accessed: August 2021.

UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0045
	Issue:	1.00
	Date:	August 27, 2021
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Will Saunders Wayne Arter BD	N/A N/A	August 27, 2021 August 27, 2021
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		August 27, 2021
Approved By:	Rob Akers Advanced Computing Dept. Manager		August 27, 2021

NEPTUNE: 2047353-TN-01
Priority Equations and Test Cases: Preconditioning Milestones M1.1 and M2.1

Sue Thorne

Version 1: January 2021; Version 2: 22 February 2021

1 Introduction

Within the NEPTUNE Programme, there are a wide variety of interesting problems [1] but, due to the length of the Preconditioning project, it is important that we prioritise a small number of equations and test cases.

2 Elliptic Problems

For elliptic problems, System 2-2 from [1] is our priority, which consists of a 2-D elliptic solver in complex geometry. BOUT++ [2] already has some test cases and Nektar++ [4] also has some suitable cases. BOUT++ has finite difference examples whilst Nektar++ uses finite and spectral/hp elements for its discretizations. There is a solver in BOUT++ that sets up a matrix problem and calls PETSc [5], and another implementation of the same problem which calls HYPRE [6]. Since BOUT++ uses finite differences and not finite or spectral elements, there are plans within the NEPTUNE Programme to implement the Nektar++ version over the next 6 months or so.

3 Hyperbolic Problems

For hyperbolic problems, System 2-3 from [1] is our priority case. There is already a BOUT++ test case called SD1D [3] that models the dynamics along the magnetic field, uses finite differences and is a matrix-free implementation that uses SUNDIALS [7]. A version of this test problem is going to be formed using Nektar++ during the next few months.

For the dynamics across the magnetic field, there are 2D problems like Hasegawa-Wakatani, which are a similar to incompressible fluid dynamics (and are a simplified version of the equations shown in Ben Dudson's talk at the

NEPTUNE Kick-Off Meeting):

$$\begin{aligned}
\frac{\partial n}{\partial t} &= -\left[\phi, n\right] + \alpha\left(\phi - n\right) - \kappa\frac{\partial \phi}{\partial z} + D_n\nabla_\perp^2 n, && (1)\\
\frac{\partial \omega}{\partial t} &= -\left[\omega, n\right] + \alpha\left(\omega - n\right) + D_\omega\nabla_\perp^2 \omega, && (2)\\
\nabla^2\phi &= \omega && (3)
\end{aligned}$$

where the equations are solved for the plasma density n and vorticity $\omega = \mathbf{b}_0 \cdot \nabla \times \mathbf{v}$, where $\mathbf{v}$ is the $E \times B$ drift velocity velocity in a constant magnetic field and $\mathbf{b}_0$ is the unit vector in the direction of the equilibrium magnetic field. The Poison bracket is represented by $[\cdot, \cdot]$.

For simplicity, if the backward Euler method is used to discretise the time derivative and we assume that n, ω and ϕ are discretized using the same discretization basis, then we obtain the following discretized, non-linear equations:

$$\begin{aligned}
0 &= M\left(\underline{n}^i - \underline{n}^{i-1}\right) + \Delta t\left(diag(L_x\underline{\phi}^i)L_z\underline{n}^i - diag(L_z\underline{\phi}^i)L_x\underline{n}^i\right. && (4)\\
&\quad \left. -\alpha M\left(\underline{\phi}^i - \underline{n}^i\right) - \kappa L_z\underline{\phi} - D_n K\underline{n}^i\right),\\
0 &= M\left(\underline{\omega}^i - \underline{\omega}^{i-1}\right) + \Delta t\left(diag(L_x\underline{\phi}^i)L_z\underline{\omega}^i - diag(L_z\underline{\phi}^i)L_x\underline{\omega}^i\right. && (5)\\
&\quad \left. -\alpha M\left(\underline{\phi}^i - \underline{n}^i\right) - D_\omega K\underline{\omega}^i\right),\\
0 &= K\underline{\phi}^i - M\omega, && (6)
\end{aligned}$$

where M is the mass matrix associated with the discretization basis, L_x and L_z are matrices that contain the discretized forms of $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial z}$, respectively, K is the discretized Laplacian matrix and $diag(\underline{x})$ denotes a diagonal matrix with the (p, p) entry equal to $\underline{x}(p)$.

The Jacobian of equations (4)-(6) with respect to the unknowns $\underline{n}$, $\underline{\omega}$ and $\underline{\phi}$ can be expressed in the following block matrix format:

$$\left[\begin{array}{ccc} A & 0 & E \\ B & C & G \\ 0 & -M & K \end{array}\right], \qquad (7)$$

where

$$\begin{aligned}
A &= M + \Delta t\left(diag(L_x\underline{\phi}^i)L_z - diag(L_z\underline{\phi}^i)L_x + \alpha M - D_n K\right), && (8)\\
B &= \alpha\Delta t M, && (9)\\
C &= M + \Delta t\left(diag(L_x\underline{\phi}^i)L_z - diag(L_z\underline{\phi}^i)L_x - D_\omega K\right), && (10)\\
E &= \Delta t\left(diag(L_z\underline{n}^i)L_x - diag(L_x\underline{n}^i)L_z - \alpha M - \kappa L_z\right), && (11)\\
G &= \Delta t\left(diag(L_z\underline{\omega}^i)L_x - diag(L_x\underline{\omega}^i)L_z\right). && (12)
\end{aligned}$$

If K, L_x, L_z and M are all sparse, then the Jacobian will be sparse.

Alternatively, we can remove $\underline{\phi}^i$ from (4) and (5) by substituting in $\underline{\phi}^i = K^{-1}\underline{\omega}^i$ from (6). The Jacobian for the reduced equations is

$$\left[\begin{array}{cc} P & R \\ Q & S \end{array}\right], \qquad (13)$$

where

$$
\begin{aligned}
P &= M + \Delta\left(diag(L_x K^{-1} M\underline{\omega}^i)L_z - diag(L_z K^{-1} M\underline{\omega}^i)L_x + \alpha M - D_n K\right), \\
Q &= \alpha\Delta t M, \\
R &= \Delta t\left(diag(L_z\underline{n}^i)L_x K^{-1} M - diag(L_x\underline{n}^i)L_z K^{-1} M - \alpha M K^{-1} M \right. \\
&\quad \left. -\kappa L_z K^{-1} M\right), \\
S &= M + \Delta t\left(diag(L_x K^{-1} M\underline{\omega}^i)L_z + diag(L_z\underline{\omega}^i)L_x K^{-1} M - \alpha M K^{-1} M \right. \\
&\quad \left. -D_\omega K\right).
\end{aligned}
$$

Note that the inverse of the elliptic matrix K is normally a dense matrix, particularly for the finite and spectral element discretizations of interest to this project. Hence, for the discretizations of interest, the Jacobian of the reduced system is not sparse. Therefore, one of the questions is whether to (a) use the reduced system but have a dense Jacobian, or (b) treat the elliptic problem as a constraint but have a sparse, larger and more ill-conditioned Jacobian. Option (a) is normally done but experiments have also been done in BOUT++ (b) in order to use PETSc's matrix coloring to extract an approximate matrix for preconditioning from our matrix-free code. Nektar++ also has an implementation of the Hasegawa-Wakatani problem.

4 General Comments

When possible, Nektar++ examples should have higher priority than BOUT++ due to the expectation that finite element and spectral discretizations will be the method of choice for future simulations. In addition, we should expect adaptive hp-refinement to be used. Nektar++ is moving towards matrix-free implementations. For Nektar++, we need to keep in contact with David Moxey.

5 Acknowledgements

I would like to thank Ben Dudson for his help in prioritising the equations and test cases.

References

[1] W. Arter. Equations for NEPTUNE Proxyapps, 2020.

[2] B. Dudson et al. BOUT++ v4.3.2, March 2020.

[3] B. Dudson et al. Sd1d: One-dimensional plasma-neutral simulation for sol and divertor studies. `https://github.com/boutproject/SD1D`, 2020.

[4] D. Moxey et al. Nektar++ website. `https://www.nektar.info`, 2020.

[5] Satish Balay et al. PETSc Web page. https://www.mcs.anl.gov/petsc, 2019.

[6] Lawrence Livermore National Laboratory. HYPRE: Scalable Linear Solvers and Multigrid Methods. https://computing.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods/software.

[7] Lawrence Livermore National Laboratory. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. https://computing.llnl.gov/projects/sundials.

NEPTUNE Technical Report 2047353-TN-02
Deliverables 1.1, 2.1 and 3.1

Linear systems of equations and preconditioners relating to the NEPTUNE Programme

A brief overview

Authors: V. Alexandrov, A. Lebedev, E. Sahin, S. Thorne

April 22, 2021

Contents

Chapter 1

Landscape of Preconditioners

1.1 Motivational Theory

Intermediate- and large-scale linear systems

$$A\vec{x} = \vec{b} \tag{1.1}$$

are most commonly solved using iterative methods such as

- (symmetric) successive over-relaxation ((S)SOR),
- Jacobi over-relaxation (JOR),
- conjugate gradient (CG),
- biconjugate gradient stabilized (BiCGstab),
- conjugate gradient squared (CGS),
- minimal residual method (MINRES),
- generalized minimal residual method (GMRES),
- quasi-minimal residual (QMR) and
- transpose-free QMR (TFQMR),

see [1], [8], [12], [24] or any university-level introduction to numerical mathematics, such as [30]. These methods do not compute the inverse system matrix A^{-1} or factors of A but approximate the solution via an iterative method of the form

$$\vec{x}_k = \vec{x}_{k-1} + \vec{s}_k \ . \tag{1.2}$$

Their convergence rates are typically bounded from above by expressions involving the condition number of the system matrix A, which is given here for the 2-norm:

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 \equiv \frac{\max_{\lambda_i \in \sigma(A)} |\lambda_i|}{\min_{\lambda_j \in \sigma(A)} |\lambda_j|} \ , \tag{1.3}$$

where $\sigma(A)$ denotes the spectrum of A. In general, the larger the value of the condition number, the slower the rate of convergence but other characteristics such as the eigenvalues being highly clustered into a few groups can significantly reduce the number of iterations required to achieve the desired level of accuracy. The above immediately shows that matrices which are almost singular, i.e., for which an eigenvalue λ_k exists s.t. $|\lambda_k| \approx 0$ the condition number becomes very large, are likely to have slow convergence. For example, this occurs when a finite-element discretization of a PDE is performed with very high spatial resolution or with almost degenerate finite elements.

Table 1.1: Summary of iterative methods with their properties and any preconditioner requirements. Here "s.p.d." denotes a symmetric positive definite matrix.

Iterative Method	Symmetric A	Non-symmetric A	Other requirements on A	Preconditioner requirements
SOR	Yes	Yes	diag(A) non-singular	N/A
SSOR	Yes	No	diag(A) non-singular	N/A
JOR	Yes	Yes	diag(A) non-singular	N/A
CG	Yes	No	positive definite	s.p.d.
BiCGSTAB	Yes	Yes	None	None
CGS	Yes	Yes	None	None
MINRES	Yes	No	None	s.p.d.
GMRES	Yes	Yes	None	None
QMR	Yes	Yes	None	None
TFQMR	Yes	Yes	None	None

To speed-up the iteration, one can use left and/or right preconditioners P_L and P_R, respectively, and solve the equivalent system

$$P_L A P_R \vec{y} = P_L \vec{b} \quad \text{where} \quad \mathrm{x} = \mathrm{P_R}\tilde{\mathrm{y}}, \tag{1.4}$$

and choose preconditioners for which $P_L A P_R$ has lower condition number than A or clusters the eigenvalues into a few groups such that the number of iterations is significantly reduced, and for which calculating the action of P_L and P_R multiplied by a vector is relatively cheap to initialise and then compute. For simplicity within this report, we will assume that one of P_L or P_R is the identity matrix and we will refer to the other preconditioner as P.

When considering a preconditioner for a given problem it is beneficial to keep in mind what category of PDE has given rise to the linear system at hand as well as the structure properties of the system matrix A. The latter are especially important if methods relying on the symmetry and positive definiteness of the system matrix A are employed (e.g., CG iteration). In such a case, usage of a preconditioner that is not symmetric and positive definite *may* slow the convergence of the iterative method or result in the method breaking down. We provide a summary of the different methods mentioned in Table 1.1.

1.2 Preconditioner Classes

1.2.1 Scalings

A simple way to reduce the condition number of a matrix is to scale its rows/columns to be of approximately equal magnitude w.r.t. a given norm [24],[1]. Generally, the effectiveness of scaling preconditioners can be neglected when comparing them to the other classes listed below. They should, nevertheless, not be neglected, given that they are extremely cheap to compute and can be beneficial in cases where the computation of the matrix-vector product is highly susceptible to round-off errors (i.e., when using mixed precision computations).

1.2.1.1 Point-Jacobi

The simplest preconditioner is the so-called point-Jacobi preconditioner [30], [17], [8]. It is defined as

$$P := \mathrm{diag}(A)^{-1} \quad . \tag{1.5}$$

Due to its simplicity the preconditioner can be directly computed and applied to the iterations of the chosen method. A further benefit of the simplicity is the trivial parallelizability (there are no data-dependencies). The simplicity comes at the cost of effectiveness, this preconditioner will generally only slightly reduce the number of iterations required to achieve convergence.

1.2.1.2 Norm-based scaling

If one chooses

$$P = \mathrm{diag}\left(\left\{\frac{1}{d_{ii}}\right\}_{i=1}^{n}\right) := \mathrm{diag}\left(\left\{\frac{1}{\|\vec{a}_i\|_m}\right\}_{i=1}^{n}\right) \tag{1.6}$$

where $\vec{a}_i$ can be either the i-th row of the matrix A or its i-th column and $\|\cdot\|_m$ is the m norm of a vector, then the preconditioner can be trivially inverted for a direct application to the iteration vector. Such preconditioners are generally referred to as row-/column-scalings. Formally it can be proven that

$$\kappa_\infty(\tilde{P}A) \leq \kappa_\infty(PA) \tag{1.7}$$

for any scaling P if $\tilde{P}$ is computed using the 1 norm ($\|\vec{a}\|_1 = \sum_j |a_j|$) in (1.6), i.e., the preconditioner is an *optimal* scaling.

1.2.2 Incomplete Factorizations

Factorization methods that decompose a matrix into a product of matrices, i.e., $A = LU$, can be used as a basis to derive preconditioners. Note that the factors of the matrix *will generally become dense*, even if A is sparse. This is generally avoided by performing the factorization only for a pre-defined number of non-zero elements. Such a factorization is generally incomplete, hence the name of this preconditioner class. One strategy is to preserve the non-zero pattern of the matrix A, which results in zero-fill preconditioners.

Due to the sequential nature of the underlying factorization, these methods generally require a preliminary graph partitioning (element reordering) to be parallelized. The scalability of such methods is thus *limited* by the number of (strongly) connected components of the graph obtained if the matrix is interpreted as an adjacency matrix of a graph.

1.2.2.1 D-ILU

The second on the triviality scale is the D-ILU preconditioner [11]. It is based on the decomposition of the matrix A similar to the Jacobi iteration:

$$P := (D + L_A)D^{-1}(D + U_A) \quad , \tag{1.8}$$

where D is now not simply the diagonal of A but determined according to a different scheme and U_A, L_A are the strict upper/lower triangular parts of the matrix A.

1.2.2.2 ILU - Incomplete LU Decomposition

A solution of $LU\mathbf{x} = \mathbf{b}$, where L, U are obtained by LU-decomposition (Gaussian elimination) of A is equivalent to $\mathbf{x} = A^{-1}\mathbf{b}$. Here, L is a lower triangular matrix and U is an upper triangular matrix. As such one obvious choice for a preconditioner is the LU-decomposition of A. However, a full LU-decomposition may change a sparse matrix A into a dense one so we could, instead, only perform a step of the LU decomposition if and only if $a_{i,j} \neq 0$. This yields the so-called zero-fill incomplete LU factorization - in short: ILU(0).

ILU(k) denotes an ILU preconditioner with a user-defined fill-in level $k \geq 0$. Higher k correspond generally to a better approximation of the LU decomposition but result in a much denser preconditioner. As a rule $k > 3$ is seldom used [11].

ILUT(ρ,τ) is a threshold variant of ILU, in which entries are removed from the preconditioner if their magnitude falls under the threshold τ or the fraction ρ of additional values per row is exceeded.

1.2.2.3 IC - Incomplete Cholesky Decomposition

Applying the same methodology to the Cholesky decomposition for symmetric positive definite matrices, the IC(k) (incomplete Cholesky decomposition with fill-level k) factorization is obtained. Similar to the common Cholesky decomposition [1], it requires $\sim \frac{1}{2}$ as many operations to compute, as ILU and the resulting decomposition is symmetric, positive definite, making it a prime candidate for an incomplete factorization preconditioner for symmetric (positive definite) matrices and methods which rely on the symmetry of the linear operator (e.g. CG). The same caveats regarding fill-in as for ILU(k) apply.

1.2.2.4 Additive Factorization - Splitting

Methods which rely on a splitting of the matrix A into $A = L+D+R$ which are, by themselves, *not* modified are designated "splitting methods" and can roughly be assigned to the class of factorization methods. In the following, we assume that D is defined as for the Jacobi preconditioner, L is the strict lower triangular part of A and R is the strict upper triangular part of A.

While we provide the preconditioner matrices below *these are generally never computed* and an application of any splitting preconditioner corresponds to the execution of one iteration step of the corresponding iterative solver method.

1.2.2.4.1 Jacobi This preconditioner, which is also considered in Section 1.5, can be considered a splitting or scaling preconditioner, since it corresponds to the preconditioner matrix $P = D^{-1}$.

1.2.2.4.2 (Symmetric) Gauss-Seidel The preconditioner matrix of the simple Gauss-Seidel preconditioner is

$$P = (D + L)^{-1} . \tag{1.9}$$

For the symmetric form, the preconditioner matrix is

$$P = (D + R)^{-1} D (D + L)^{-1} . \tag{1.10}$$

1.2.2.4.3 (Symmetric) SOR This preconditioner is equivalent to the Successive Overrelaxation (SOR) method, resulting in the preconditioner matrix

$$P = \omega (D + \omega L)^{-1} . \tag{1.11}$$

Here ω is a parameter supplied by the user with $\omega \in (0, 2)$. Its symmetrised counterpart is given by

$$P = \omega (2 - \omega)(D + \omega R)^{-1} D (D + \omega L)^{-1} . \tag{1.12}$$

The (s)SOR preconditioners correspond to shortened ($\omega < 1$) or prolonged ($\omega > 1$) Gauss-Seidel steps and, as such, can be attempted once it has been established that the (s)GS preconditioners do not result in the desired reduction of step number. Since an iterative solution is, simply put, an update of the proposed solution $\vec{x}_k$ at step k by a correction - the residual $\vec{r}_k$ - one may introduce an importance weighting of the correction ω: $\vec{x}_{k+1} = \vec{x}_k + \omega \vec{r}_k$. Dependent upon the problem and the current approximation $\vec{x}_k$ a larger or smaller correction to $\vec{x}_k$ can be performed.

1.2.3 Approximate Inverses

Instead of computing an approximate factorization of the sparse matrix A sequentially, one may directly approximate its inverse by minimizing

$$\|I - BA\|_2 \tag{1.13}$$

w.r.t. B in one go. This class of preconditioners generally yields better to parallelisation attempts, at the cost of higher memory complexity, since (1.13) essentially factorizes into independent least-squares approximations. Note that while for the true inverse holds $A^{-1}A = I = AA^{-1}$ this does not hold for approximate inverses, i.e. a right-inverse obtained using $\|I - AB\|_2$ will generally differ from a left inverse, obtained from $\|I - BA\|_2$.

1.2.3.1 SPAI - SParse Approximate Inverse

The method enlarges the non-zero pattern of the approximant B dynamically until the minimization problem is solved to within a provided tolerance. Minimisation of the residual results in the method providing robust preconditioners at the cost of being time-consuming. This approach does not guarantee that the approximate inverse B of a symmetric matrix will be symmetric.

1.2.3.2 FSAI - Factorized Sparse Approximate Inverse

This variant of SPAI does not approximate $B = A^{-1}$ directly but rather the Cholesky factors of B, i.e., L in $A^{-1} \approx L^t L$. Usage of Cholesky factors imposes the same restrictions on the matrix A as the (incomplete) Cholesky decomposition 1.2.2.3 - A has to be symmetric positive definite. If A is s.p.d., then FSAI is well-defined, the converse - in general - does not hold.

1.2.3.3 AINV - Approximate INVerse

Similar to FSAI this method approximates the inverses of triangular factors which, if fully computed, transform the matrix A into a diagonal matrix: $L^{-1}AU^{-1} = D$.

1.2.3.4 MCMCMI - Markov Chain Monte Carlo Matrix Inversion

This method computes a sparse approximate inverse approximation via a random walk on the graph defined by interpreting A as an adjacency matrix. We discuss this further in Section 1.2.5.2.

1.2.4 Multigrid Methods

Similar to the splitting preconditioners, multigrid preconditioners originate from multigrid solution methods. The latter are intended to accelerate the convergence of, e.g., Gauss-Seidel iterations for large systems by essentially coarsening the solution vector $\vec{x}$ (e.g., by selecting only every second element), computing a solution with $\vec{x}$ as the right-hand side vector $\vec{b}$ of a smaller linear system and finally interpolating the coarse solution onto the original solution and updating the original $\vec{x}$ [31].

Originally, the methods were conceived for the solution of elliptic PDE where highly oscillatory components of the residual solution can be damped by solving computing a solution on a coarser mesh and updating the solution on the finer mesh. *Algebraic* multigrid methods (AMG) abstract the geometric picture of a mesh away and attempt to replicate the procedure given only the system matrix A. Multigrid methods vary by the choice of the coarsening and interpolation operators as well as of the coarsening/refinement cycles. These methods are very well suited for linear systems resulting from the discretization of elliptic PDE. They are especially suited for approaches where iterative mesh refinement is utilised. Furthermore, these methods scale well, provided the utilised solver is well-parallelised. In the case of a strongly heterogeneous system (i.e., multiscale system) or of strongly irregular meshes (e.g., mesh refinement at a sharp tip) these methods are known to fail.

1.2.4.1 *l*AIR

Classical AMG method variants are available for both symmetric and nonsymmetric problems although the former are more widely known and used. The *l*AIR preconditioner is a variation on classical AMG for nonsymmetric matrices [22]. The method is based on a local approximation to an ideal restriction operator, which is coupled with F-relaxation. For a given mesh with

vertex set V, the set V is partitioned into F-points and C-points, where C-points represent vertices on the coarse grid. The matrix A can then be symbolically ordered into the following block form:

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix},$$

where A_{ff} corresponds to the F-points. F-relaxation improves the solution at the F-points and this accuracy is then distributed at the C-points via the coarse-grid correction (ideal restriction):

$$R = \begin{pmatrix} -A_{cf}A_{ff}^{-1} & I \end{pmatrix}.$$

It has been shown to be a robust solver for various discretizations of the advection-diffusion-reaction equation in regimes ranging from purely advective to purely diffusive and including time-dependent and steady-state problems.

1.2.5 Stochastic Methods

The methods presented above are deterministic and generally touch each value of the matrix at least once during the computation. Stochastic methods aim to reduce computational costs by utilising only the "important subset" of the matrix/vector entries.

A fairly accurate interpretation of stochastic methods would be as "measurement", where a systematic error (equivalent to the tolerances for deterministic methods) is to be balanced with a stochastic measurement error. Usage of stochastic methods for the computation of preconditioners uses the fact that a preconditioner does not have to be *excellent* to be usable, but instead has to be effective and quick to compute.

1.2.5.1 SP -Stochastic Projection

The basic idea is fairly simple and the implementation follows roughly equation (2.13) of [34]. The main idea is to project the solution vector successively and orthogonally onto arbitrarily chosen subspaces of the row-space of the matrix until the accumulated effect leads the iteration into the subspace of the true solution. An obvious extension to block-projections has been mentioned in [34] with the iteration step given as follows:

$$\vec{x}_{k+1} = \vec{x}_k + A_i^t \left(A_i A_i^t\right)^{-1} \left(\vec{b}_i - A_i \vec{x}_k\right) . \tag{1.14}$$

Here A_i is a randomly selected block of rows of the matrix and $\vec{b}_i$ the corresponding subset of entries of the right-hand-side vector of

$$A\vec{x} = \vec{b} . \tag{1.15}$$

The intuitive simplicity of this approach is paid for by its performance. Furthermore, the computation of a matrix inverse in each step is required. Depending on the block size of the computation of said inverse, or a solution of a dense system, may incur a significant cost.

1.2.5.2 MCMCMI - Markov Chain Monte Carlo Matrix Inversion

This method computes a sparse approximate inverse by performing a random walk on the graph defined by interpreting the matrix A as an adjacency matrix. The entries of the inverse are computed by utilising the Neumann series:

$$A^{-1} = \sum_{i=0}^{\infty} (I - A)^i . \tag{1.16}$$

This requires $\rho(A) < 1$ in general, but by proper scaling can be used even when this condition is not fulfilled. The benefit of this method is that it does not require the matrix A to be explicitly known and the computational cost of computing *either one row of the inverse or one element of the solution vector* scales as $\mathcal{O}(NT)$, where N is the number of Markov chains and T the mean length of a chain.

1.3 Further Remarks

1.3.1 On Parallelism

As has been remarked above, many of the factorization methods require a prior graph-partitioning to restructure the system matrix - ideally in a block-diagonal form. The Gauss-Seidel preconditioning iteration, for instance, carries an explicit data dependency. This can be circumvented by usage of so-called wave-front parallelization [13] but is, as a rule, not part of the solver implementation.

Approximate inverses are generally much more amenable to parallelization due to the aforementioned independence of the least-squares problems that need to be solved. Stochastic methods such as MCMCMI fall into the same category.

1.3.1.1 Domain decomposition

The decomposition of the computational simulation domain with a local ordering of the nodes of the mesh improves the feasibility of factorization methods by reducing the effects of data dependencies and concentrating matrix entries relevant for the local physical domain on the local computational domain. This in turn simplifies the partitioning of the system matrix into block-diagonal form (with or without overlap). An additive Schwarz preconditioner performs a factorization on the (overlapping) blocks in parallel, with an additive averaging on the overlaps.

1.3.2 On Matrices

Most of the aforementioned methods can be used matrix-free, since they generally only require the ability to determine the elements of the matrix and compute a matrix-vector product. Incomplete factorizations with fill-in (ILU(k), IC(k)) are less amenable to usage with matrix-free methods due to the need to compute the fill indices of the matrix elements and to create a factorization with a non-zero pattern larger than the original system matrix.

Scalings and approximate inverses are more amenable to use with matrix-free methods due to the only requirement being the computation of the matrix elements.

1.4 Summary of Preconditioners

In Table 1.2, we summarise the preconditioners as well as their prerequisites and limitations. We distinguish between regular (i.e., non-singular, invertible) matrices and general matrices. The latter do not have to be square, in which case one computes an approximate pseudo-inverse, rather than an approximate true inverse.

The column "CG usable" indicates whether a preconditioner computed with a given method is usable with an iterative solver which requires a symmetric, positive matrix (represented by the CG iteration), i.e., whether the method produces a symmetric, positive definite preconditioner. This information is intended as a rough guide only, since asymmetric preconditioners (e.g., as computed by SPAI or MCMCMI) may still work as intended.

Constraints on the system matrix given in the second column stem, in part, from theoretical considerations. Methods such as IC, (s)SOR, FSAI may still compute a valid preconditioner if the system matrix is indefinite but the theory guarantees that, with exact arithmetic, these methods will not fail if the matrix is symmetric and positive definite.

Table 1.2: Summary of preconditioners and their application scenarios. Here "s.p.d." denotes a symmetric positive definite matrix, "regular" an invertible matrix, "general" a general (non-square) matrix and "d.d." a dominant diagonal. "GP" abbreviates "graph partitioning" and "WP" "wavefront parallelization"[13] .

Preconditioner	System matrix	PDE type	Matrix-free	A required explicitly	parallelizable	CG usable
Jacobi	$0 \notin \text{diag}(A)$, regular	general	Yes	No	Yes	Yes
Scaling	regular	general	Yes	No	Yes	Yes
D-ILU	$0 \notin \text{diag}(A)$, regular	general	Yes	No	via GP	Yes
ILU($k \geq 0$)/ILUT(ρ, τ)	regular	general	No	Yes	via GP	No
IC($k \geq 0$)	s. p. d.	general (elliptic)	No	Yes	via GP	Yes
(s)GS	regular, d.d.	general	Yes	No	via WP	sGS only
(s)SOR	s. p. d.	general (elliptic)	Yes	No	via WP	sSOR only
SPAI	general	general	No	No	Yes	No
FSAI	s. p. d.	general (elliptic)	No	No	Yes	Yes
AINV	regular	general	No	No	limited	No
AMG	regular	elliptic, hyperbolic, parabolic	Yes	No	Yes	Yes
lAIR	regular	elliptic, parabolic	Yes	No	undetermined	-
SP	general	general	Yes	No	Yes	No
MCMCMI	general	general	No	No	Yes	No

Chapter 2

Application Cases

2.1 Introduction

Due to the limited duration of the NEPTUNE Preconditioning Project, the variety of problems and test cases available within the NEPTUNE programme [2] has to be reduced to a manageable level. In the following we present the equations and test cases that will be considered within the given project.

2.2 Elliptic Problems

For elliptic problems, System 2-2 from [2] is our priority, which consists of a 2-D elliptic solver in complex geometry. BOUT++ [9] already has some test cases and Nektar++ [35] also has some suitable cases. BOUT++ has finite difference examples whilst Nektar++ uses finite and spectral/*hp* elements for its discretizations. There is a solver in BOUT++ that sets up a matrix problem and calls PETSc [29], and another implementation of the same problem which calls HYPRE [10]. Since BOUT++ uses finite differences and not finite, high order, elements, there are plans within the NEPTUNE Programme to implement the Nektar++ version over the next 6 months or so.

2.3 Hyperbolic Problems

For hyperbolic problems, System 2-3 from [2] is our priority case. There is already a BOUT++ test case called SD1D [9] that models the dynamics along the magnetic field, utilises finite differences and is a matrix-free implementation that uses SUNDIALS [40]. A version of this test problem is going to be set up using Nektar++ during the next few months. For the dynamics across the magnetic field, there are 2D problems like Hasegawa-Wakatani, which are a similar to incompressible fluid dynamics:

$$\frac{\partial n}{\partial t} = -\{\phi, n\} + \alpha(\phi - n) - \kappa\frac{\partial \phi}{\partial z} + D_n\nabla_\perp^2 n \tag{2.1a}$$

$$\frac{\partial \omega}{\partial t} = -\{\omega, n\} + \alpha(\omega - n) + D_\omega\nabla_\perp^2 \omega \tag{2.1b}$$

$$\nabla^2\phi = \omega\ . \tag{2.1c}$$

Here n is the plasma number density, $\omega := \vec{b}_0\cdot\nabla\times\vec{v}$ is the vorticity with $\vec{v}$ being the $\vec{E}\times\vec{B}$ drift velocity in a constant magnetic field and $\vec{b}_0$ is the unit vector in the direction of the equilibrium magnetic field. The operator $\{\cdot,\cdot\}$ in (2.1) is the Poisson bracket.

Let us assume for simplicity, that n, ω, ϕ are discretized using the same basis in space and an implicit Euler method is used for the time evolution, then the following set of non-linear equations is obtained from (2.1):

$$\begin{aligned}
0 &= M(\vec{n}_i - \vec{n}_{i-1}) + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z\vec{n}_i - \text{diag}\left(L_z\vec{\phi}_i\right) L_x\vec{n}_i \right. \\
&\quad \left. - \alpha M\left(\vec{\phi}_i - \vec{n}_i\right) - \kappa L_z\vec{\phi}_i - D_n K\vec{n}_i\right) && \text{(2.2a)} \\
0 &= M\left(\vec{\omega}_i - \omega_{i-1}\right) + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z\vec{\omega}_i - \text{diag}\left(L_z\vec{\phi}_i\right) L_x\vec{\omega}_i \right. \\
&\quad \left. - \alpha M\left(\vec{\phi}_i - \vec{n}_i\right) - D_\omega K\vec{\omega}_i\right) && \text{(2.2b)} \\
0 &= K\vec{\phi}_i - M\vec{\omega}_i \ , && \text{(2.2c)}
\end{aligned}$$

where K, M are the stiffness and mass matrices respectively and L_x, L_z are transport matrices and represent the discretized versions of ∂_x, ∂_z [6]. Note that the first two terms in the Δt bracket in the first two equations (the terms with two L operators) are straightforward discretizations of the Poisson bracket $\{f, g\} = \partial_x f \partial_z g - \partial_z f \partial_x g$ where the recasting of, e.g. $L_z\vec{\phi}$, is necessary to ensure that the result of $\text{diag}\left(L_z\vec{\phi}_i\right) L_x\vec{n}_i$ is again a vector (a discretized function).

Since the equations are non-linear they have to be solved e.g. via Newton's method, which requires the Jacobian of the system. The latter has the following form:

$$J := \begin{bmatrix} A & 0 & E \\ B & C & G \\ 0 & -M & K \end{bmatrix} \ , \tag{2.3}$$

where the constituent matrices are the following

$$\begin{aligned}
A &= M + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z - \text{diag}\left(L_z\vec{\phi}_i\right) L_x + \alpha M - D_n K\right) && \text{(2.4a)} \\
B &= \alpha \Delta t M && \text{(2.4b)} \\
C &= M + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z - \text{diag}\left(L_z\vec{\phi}_i\right) L_x - D_\omega K\right) && \text{(2.4c)} \\
E &= \Delta t \left(\text{diag}\left(L_z\vec{n}_i\right) L_x - \text{diag}(L_x\vec{n}_i) L_z - \alpha M - \kappa L_z\right) && \text{(2.4d)} \\
G &= \Delta t \left(\text{diag}(L_z\vec{\omega}_i) L_x - \text{diag}(L_x\vec{\omega}_i) L_z\right) \ . && \text{(2.4e)}
\end{aligned}$$

Alternatively the last equation of (2.2) can be used to eliminate $\vec{\phi}_i$ by virtue of $\vec{\phi}_i = K^{-1}M\vec{\omega}_i$. The Jacobian of the reduced system is then of the form

$$\tilde{J} := \begin{bmatrix} P & R \\ Q & S \end{bmatrix} \ , \tag{2.5}$$

with the constituent matrices

$$\begin{aligned}
P &= M + \Delta t \left(\text{diag}\left(L_x K^{-1} M\vec{\omega}_i\right) L_z - \text{diag}\left(L_z K^{-1} M\vec{\omega}_i\right) L_x + \alpha M - D_n K\right) && \text{(2.6a)} \\
Q &= \alpha \Delta t M && \text{(2.6b)} \\
R &= \Delta t \left(\text{diag}\left(L_z\vec{n}_i\right) L_x K^{-1} M - \text{diag}(L_x\vec{n}_i) L_z K^{-1} M - \alpha M K^{-1} M - \kappa L_z K^{-1} M\right) && \text{(2.6c)} \\
S &= M + \Delta t \left(\text{diag}\left(L_x K^{-1} M\vec{\omega}_i\right) L_z + \text{diag}\left(L_z\vec{\omega}_i\right) L_x K^{-1} M - \alpha M K^{-1} M - D_\omega K\right) && \text{(2.6d)}
\end{aligned}$$

As mentioned above, even if the stiffness matrix K is sparse its inverse will generally be dense. Hence, for the discretization methods of interest for the project, the Jacobian of the reduced system $\tilde{J}$ will not be sparse. Therefore, one of the questions is whether to (a) use the

reduced system but have a dense Jacobian, or (b) treat the elliptic problem as a constraint but have a sparse, larger and more ill-conditioned Jacobian. Option (a) is normally done but experiments have also been done in BOUT++ (b) in order to use PETSc's matrix coloring to extract an approximate matrix for pre- conditioning from our matrix-free code. Nektar++ also has an implementation of the Hasegawa-Wakatani problem.

2.4 Comments

When possible, Nektar++ examples should have higher priority than BOUT++ due to the expectation that finite element and spectral discretizations will be the method of choice for future simulations. In addition, we should expect adaptive hp-refinement to be used. Nektar++ is moving towards matrix-free implementations. For Nektar++, we need to keep in contact with David Moxey.

Chapter 3

Implementations

3.1 Introduction

For scientific computing, there are widely-used platforms such as (among others) PETSc/TAO [29], Trilinos, BOUT++ [9], Nektar++[35] and SUNDIALS [40] available. According to the prioritised equations in Chapters 2, there are several studies using these platforms accessible in the literature.

3.2 Elliptic Problems

As indicated in Chapter 2, the Grad-Shafranov equations is used to generate spectrally accurate magnetic fields to use in other proxyapps [2]. There are highly-scalable, multi-physics implementations for Grad-Shafranov using finite difference [20] [41] and finite element [28] methods with PETSc as well as finite element methods [32] [5] with Trilinos. To solve another equation of interest, the non-Boussinesq vorticity equation, [3] used preconditioned Conjugate Gradient with a Block–Jacobi preconditioner from PETSc. In [18], an inexact Newton-Krylov approach with PETSc was used to solve a similar problem. In addition, [4] uses a matrix-free method from PETSc to solve a non-Boussinesq formulation of polythermal ice flow. In [33], similar problems were solved with massive MPI parrallelism using Krylov methods with preconditioners (ILU(k) and block Jacobi) from PETSc. As discussed in Chapter 2, there are implementations and suitable cases available in BOUT++ [9] and Nektar++ [35]. There are also tools which use the mixture of platforms such as libMesh [16] and CoolFluiD[19] framework specialised for plasma and multi-physics simulations.

3.3 Hyperbolic Problems

As stated in Chapter 2, the hyperbolic case focuses on system 2-3 of [2], of which several related implementations and test cases exist in BOUT++ and Nektar++.

In the literature, several studies exist in which highly parallelized solvers for a 2-fluid model have been considered and implemented. In [23], the linearized two-fluid MHD equations were solved using a matrix-free Newton-Krylov method of solution in XTOR-2F (PETSc) with MPI parallelism. In [39], an MPI parallelized Vlasov Fokker-Planck (VFP) set of equations was solved (IMPACT [15]) using BICGstab as solver with an ILU preconditioner provided by PETSc. In [21], a system of multi-fluid equations was solved using GMRES with a parallel Additive Schwarz preconditioner provided by PETSc. VFP-based equations are also solved by a package introduced in [25] and based on PETSc with MPI parallelism. In addition, there is a package [38] available within Trilinos.

3.4 Other libraries of interest to the NEPTUNE Programme

Clearly, the preconditioners from PETSc and Trilinos are already highly-used and their applicability for future exascale implementations will need exploring. However, our literature searches also identified other interesting libraries libraries.

DUNE is an open-source high-performance iterative solver that has an implementation of the GenEO preconditioner [36] for strongly anisotropic elliptic partial differential equations implemented within dune-pdelab. This preconditioner has a MPI implementation: good weak and strong scalability demonstrated on ARCHER. We note that there is a version of the GenEO preconditioner is available in PETSc.

HYPRE [10] is a library of scalable linear solvers and multigrid methods from Lawrence Livermore National Laboratory. MPI implementations are provided.

HSL_MI20 is an implementation of an algebraic multigrid preconditioner for nonsymmetric systems from the HSL library [14]. Unfortunately, parallelism is only provided through the use of parallelized Level 3 BLAS subroutine calls and it is not suitable for matrix-free/operator only provision of A.

MUMPS [26] is a Fortran 95 library that utilizes MPI and OpenMP to solve large sparse systems via direct methods and would be a good option for comparing iterative methods with direct methods. The library is developed within a consortium of people from CERFACS, ENS Lyon, INPT(ENSEEIHT)-IRIT, Inria, Mumps Technologies and the University of Bordeaux.

PaStiX (Parallel Sparse matriX package) [27] is a scientific library that provides a high performance parallel solver for very large sparse linear systems based on direct methods. The library can also be used to form preconditioners. As well as using MPI and OpenMP, the most recent versions of the library also support the use of GPUs.

The STRUMPACK [37] library provides linear algebra routines and linear system solvers for sparse and for dense rank-structured linear systems. The library has Fortran and C interfaces and can also be used to compute incomplete LU factorization-based preconditioners. Parallization is provided via MPI and OpenMP but the preconditioner routines do not currently support GPUs.

We also direct the reader to Jack Dongarra's list of freely available software for linear algebra [7], which provides a list of software and associated attributes, and is periodically updated.

Bibliography

[1] P. Arbenz, O. Chinellato, M. Gutknecht, and M. Sala. Software for Numerical Linear Algebra. online, 2006.

[2] W. Arter and R. Akers. ExCALIBUR equations for NEPTUNE Proxyapps. techreport, UK Atomic Energy Authority, 2020.

[3] B. Bigot, T. Bonometti, L. Lacaze, and O. Thual. A simple immersed-boundary method for solid–fluid interaction in constant-and stratified-density flows. *Computers & Fluids*, 97:126–142, 2014.

[4] J. Brown, M. G. Knepley, D. A. May, L. C. McInnes, and B. Smith. Composable linear solvers for multiphysics. In *2012 11th International Symposium on Parallel and Distributed Computing*, pages 55–62. IEEE, 2012.

[5] P. Daniel, A. Ern, I. Smears, and M. Vohralík. An adaptive hp-refinement strategy with computable guaranteed bound on the error reduction factor. *Computers & Mathematics with Applications*, 76(5):967–983, 2018.

[6] P. Deuflhard and M. Weiser. *Adaptive Numerical Solution of PDEs*. deGruyter, 2013.

[7] J. Dongarra. Freely Available Software for Linear Algebra, 2018. URL `http://www.netlib.org/utk/people/JackDongarra/la-sw.html`.

[8] J. Dongarra, R. Barett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 2014.

[9] B. Dudson, P. Hill, and J. Parker. BOUT++. online repository, 2020. URL `http://boutproject.github.io`.

[10] R. Falgout, A. Barker, T. Kolev, R. Li, S. Osborn, D. Osei-Kuffuor, V. P. Magri, and J. Schroeder. HYPRE: Scalable Linear Solvers and Multigrid Methods. online, 2017. URL `https://computing.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods`.

[11] M. Ferronato. Preconditioning for Sparse Linear Systems at the Dawn of the 21st Century: History, Current Developments, and Future Prospects. *ISRN Applied Mathematics*, 2012:49, October 2012. doi: 10.5402/2012/127647.

[12] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU press, 2013.

[13] G. Hager and G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. Chapman and Hall/CRC, 2011. Asian Reprint.

[14] HSL Development Team. The HSL Mathematical Software Library, 2015. URL `https://www.hsl.rl.ac.uk`.

[15] R. Kingham and A. Bell. An implicit Vlasov–Fokker–Planck code to model non-local electron transport in 2-D with magnetic fields. *Journal of Computational Physics*, 194(1):1–34, 2004.

[16] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22(3-4):237–254, 2006.

[17] D. Kressner. *Lecture notes for the course on Numerical Methods held by Daniel Kressner in the spring semester 2010 at the ETH Zurich. (Numerische Methoden Vorlesungsskript zur Veranstaltung Numerische Methoden gehalten von Daniel Kressner im FS 2010 an der ETH Zürich)*. ETH Zürich, May 2010.

[18] M. Kumar and G. Natarajan. Unified solver for thermobuoyant flows on unstructured meshes. In *Fluid Mechanics and Fluid Power–Contemporary Research*, pages 569–580. Springer, 2017.

[19] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle, and S. Poedts. The COOLFluiD framework: design solutions for high performance object oriented scientific computing software. In *International Conference on Computational Science*, pages 279–286. Springer, 2005.

[20] S. Liu, Q. Tang, and X.-Z. Tang. A parallel cut-cell algorithm for the free-boundary Grad-Shafranov problem. *arXiv preprint arXiv:2012.06015*, 2020.

[21] Y. G. Maneva, A. A. Laguna, A. Lani, and S. Poedts. Multi-fluid modeling of magnetosonic wave propagation in the solar chromosphere: effects of impact ionization and radiative recombination. *The Astrophysical Journal*, 836(2):197, 2017.

[22] T. Manteuffel, J. Ruge, and B. Southworth. Nonsymmetric algebraic multigrid based on local approximate ideal restriction (*l*air). *SIAM Journal on Scientific Computing*, 6(40):4105–4130, 2018.

[23] A. Marx and H. Lütjens. Hybrid parallelization of the XTOR-2F code for the simulation of two-fluid MHD instabilities in tokamaks. *Computer Physics Communications*, 212:90–99, 2017.

[24] A. Meister. *Numerics of systems of linear equations (Numerik Linearer Gleichungssysteme)*. Springer Spektrum, 5 edition, 2014. doi: 10.1007/978-3-658-07200-1. With MATLAB exercises.

[25] S. Mijin, A. Antony, F. Militello, and R. J. Kingham. SOL-KiT—Fully implicit code for kinetic simulation of parallel electron transport in the tokamak Scrape-Off Layer. *Computer Physics Communications*, 258:107600, 2021.

[26] MUMPS Development Team. MUMPS: MUltifrontal Massively Parallel sparse direct Solver, 2020. URL `http://mumps.enseeiht.fr/index.php?page=home`.

[27] PaStiX Development Team. PaStiX: Parallel Sparse matriX package, 2019. URL `http://pastix.gforge.inria.fr/files/README-txt.html`.

[28] Z. Peng, Q. Tang, and X.-Z. Tang. An Adaptive Discontinuous Petrov–Galerkin Method for the Grad–Shafranov Equation. *SIAM Journal on Scientific Computing*, 42(5):B1227–B1249, 2020.

[29] PETSc Development Team. PETSc Web page, 2019. URL `https://www.mcs.anl.gov/petsc`.

[30] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, 2002.

[31] R. Rabenseifner, A. Meister, et al. Iterative Solvers and Parallelization - Course material for HLRS Course 2016-ITER-S, 2016.

[32] N. V. Roberts, D. Ridzal, P. B. Bochev, and L. Demkowicz. A toolbox for a class of discontinuous Petrov-Galerkin methods using Trilinos. *Technical Report SAND2011-6678, Sandia National Laboratories*, 2011.

[33] A. L. Rossa and A. L. Coutinho. Parallel adaptive simulation of gravity currents on the lock-exchange problem. *Computers & Fluids*, 88:782–794, 2013.

[34] K. Sabelfeld and N. Loshchina. Stochastic iterative projection methods for large linear systems. *Monte Carlo Methods and Applications*, 2010. doi: 10.1515/mcma.2010.020.

[35] S. Sherwin, M. Kirby, C. Cantwell, and D. Moxey. Nektar++. online, 2021. URL `https://www.nektar.info`.

[36] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl. Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps. *Numerische Mathematik*, 1(126):741–770, 2014.

[37] STRUMPACK Development Team. STRUMPACK - STRUctured Matrix PACKage, 2021. URL `https://portal.nersc.gov/project/sparse/strumpack/v5.0.0/index.html`.

[38] Trilinos Project Team. Stokhos Package for Intrusive Stochastic Galerkin Methods, 2021. URL `https://trilinos.github.io/stokhos.html`.

[39] B. Williams and R. Kingham. Hybrid simulations of fast electron propagation including magnetized transport and non-local effects in the background plasma. *Plasma Physics and Controlled Fusion*, 55(12):124009, 2013.

[40] C. S. Woodward, D. R. Reynolds, A. C. Hindmarsh, D. J. Gardner, and C. J. Balos. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. online, 2021. URL `https://computing.llnl.gov/projects/sundials`.

[41] P. Zhu, C. Sovinec, C. Hegna, A. Bhattacharjee, and K. Germaschewski. Nonlinear ballooning instability in the near-Earth magnetotail: Growth, structure, and possible role in substorms. *Journal of Geophysical Research: Space Physics*, 112(A6), 2007.

Science and Technology Facilities Council

Implicit-factorization preconditioners for NEPTUNE Programme

Technical Report 2047353-TN-03

Maksims Abaļenkovs* Vassil Alexandrov* Anton Lebedev* Emre Sahin*
Sue Thorne**

July 2021

1 Introduction

In simulations relating to plasma physics, and more generally, the dominate component in terms of simulation time is normally the time to solve the underlying linear systems

$$Ax = b.$$

In this report, we will assume that $A \in \mathbb{R}^{n\times n}$ is non-singular, the right-hand side $b \in \mathbb{R}^n$ is provided and we wish to compute an (approximate) solution $x \in \mathbb{R}^n$. These systems are usually solved via an iterative method such a Krylov subspace method. Typically, the nature of these systems mean that a large number of iterations are required to reach the desired level of accuracy. To reduce the number of iterations, we aim to choose a preconditioner $P \in \mathbb{R}^{n\times n}$ such that applying the iterative method to the equivalent system

$$P^{-1}Ax = P^{-1}b$$

results in a reduction in the number of iterations and the time to set-up P and to solve with P during each iterations is such that there is an overall reduction in solution time. Note, in the above, we have described left preconditioning. For right preconditioning, one is solving the equivalent system

$$AP^{-1}y = b, \quad x = P^{-1}y.$$

It is also possible to use a combination of left and right preconditioners together:

$$P_1^{-1}AP_2^{-1}y = P_1^{-1}b, \quad x = P_2^{-1}y.$$

2 Software for Plasma Physics Modelling: Overview

In the following, we describe our use of BOUT++ and Nektar++, which have been identified by UKAEA to be the modelling packages of interest for the NEPTUNE Project. As part of the descriptions, we include how preconditioners are currently incorporated into these libraries.

*The authors are with the Hartree Centre, STFC Daresbury Laboratory, Sci-Tech Daresbury, Keckwick, Daresbury, Warrington, WA4 4AD, UK. Email contact: maksims.abalenkovs@stfc.ac.uk

**Sue Thorne is with the STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK. Email contact: sue.thorne@stfc.ac.uk

2.1 BOUT++

BOUT++ [5] is very much designed for matrix-free calculations, which use external packages PETSc or SUNDIALS to solve the more complex problems. In themselves, these external packages use matrix-free implementations by default. This means that custom preconditioners either need to be developed in a matrix-free, operator-based manner and provided via BOUT++ or, for preconditioners that can act in a matrix-free manner, but require access to the action of the linear system being solved, it would need implementing within the under-lying library.

As part of this project, we used a nonlinear diffusion test case that simulates the effects of heat conduction in a plasma. This test case is provided as part of BOUT++. At the moment, BOUT++ relies on the PETSc package for the preconditioning work within this test case. There are multiple ways to execute the heat conduction example. In the simplest possible form the code is launched with

```
./diffusion-nl
```

IMEX-BDF2 multistep scheme is launched with

```
./diffusion-nl solver:type=imexbdf2
```

Preconditioning is enabled with the following flag

```
./diffusion-nl solver:use_precon=true
```

Finally, the Jacobian colouring with the IMEX-BDF2 is enabled with

```
./diffusion-nl solver:type=imexbdf2 solver:matrix_free=false
```

The end command used in the experiments is

```
./diffusion-nl solver:type=imexbdf2 solver:matrix_free=false solver:use_precon=true
```

Extracting and saving the system matrix was done via PETSc routines. In order to save the matrix `imex-bdf2.cxx` source code file was modified. The `precon` routine was supplemented with the following commands (Figure 1):

```
1:  PetscViewer viewer;
2:  PetscViewerASCIIOpen(MPI_COMM_WORLD, "A.mtx", &viewer);
3:  PetscViewerPushFormat(viewer, PETSC_VIEWER_ASCII_MATRIXMARKET);
4:  MatView(Jmf, viewer);
```

Figure 1: C++ source code for system matrix extraction

In this report, we also consider the SD1D test case [4], which uses BOUT++ to simulate a plasma fluid in one dimension (along the magnetic field) that interacts with a neutral gas fluid. Unlike the nonlinear diffusion example, this test case provides a preconditioner as part of the model in an operator-based manner.

2.2 Nektar++

Nektar++ [2, 6] is designed to provide discretisation and solution of partial differential equations using the spectral/hp element method.

Nektar++ accepts `*.xml` or similar formats as inputs, where the input file provides finite-element mesh and the specifications to solve the specific PDE problem. The specification of the mesh format within Nektar++ is hierarchically defined as: 1D edges as connection of vertices, 2D faces as bounding edges and 3D elements as bounding faces. Nektar++'s MultiRegions library derives mapping from these meshes and uses these mappings for different Galerkin projection methods to assembly a global linear system. Afterwards, constructed global linear system may be solved by using direct Cholesky factorisation or iterative preconditioned conjugate gradient. Nektar++ provides range selection of the preconditioners such as classical Jacobi preconditioner, coarse-space preconditioner, block preconditioner and low-energy preconditioner. It is also providing interface to use PETSc for additional solvers. Nektar++ evaluates L_2 and L_{inf} errors against the provided exact solution.

Due to the structure of the MCMCMI algorithm, it is not feasible to adapt it as a preconditioner module for the Nektar++ in the timeline of the project according to the steps described in [2]. Therefore, Nektar++ is edited accordingly to extract full system matrices as Matrix Market format `*.mtx` to use it separately in the MCMCMI algorithm.

Nektar++ provides Advection-Diffusion-Reaction solver to solve partial differential equations of the form (1) in either discontinuous or continuous projections of the solution field [6].

$$\alpha \frac{\partial u}{\partial t} + \lambda u + v \nabla u + \epsilon \nabla \cdot (D \nabla u) \quad = \quad f \tag{1}$$

However, it is not possible to construct linear systems for the discontinuous Galerkin. Hence, application cases with continuous Galerkin were chosen from 'Nektar++/Solvers/ADRSolver/Tests' accordingly to the prioritised equations as described in [1].

2.3 Testing Environment

Numerical experiments were run on the SKL (Skylake) nodes of the Scafell Pike system which consists of nodes fitted with 2x XEON gold E5-6142 v5 processors resulting in 32 cores per node and, due to HyperThreading, 64 threads per node.

3 Numerical Results for Sparse Approximate Inverse Preconditioners

MCMCMI experiments were performed on four system matrices extracted from the BOUT++ software. These matrices were created for one and the same one-dimensional "Non-linear diffusion" test problem called `diffusion-nl`. The main BOUT++ source code was modified to enforce PETSc, powering the solution of `diffusion-nl` problem, to output system matrices into the Matrix Market `*.mtx` format. Matrix orders of the extracted BOUT++ system matrices were $n = 128, 512, 2048, 8192$. The ever-increasing system matrices were obtained by increasing the resolution over the y axis. Unfortunately, it was not possible to extend the problem even further since PETSc started to crash for $ny > 8192$.

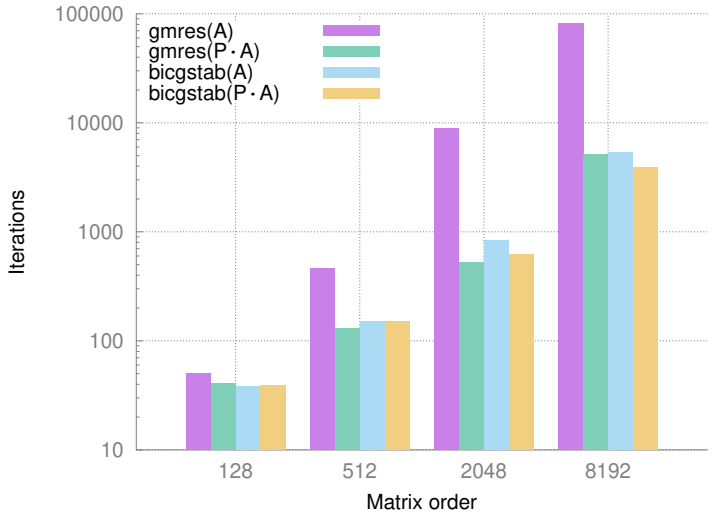

Figure 2: **Matrix order vs GMRES and BiCGStab iteration steps, BOUT++.**

In Figure 2, we compare the matrix order vs GMRES and BiCGStab iteration steps for the BOUT++ examples. In the case of GMRES method and small matrices ($n = 128, 512$), the MCMCMI preconditioner P decreases the number of iterations required for the linear system solution by 18% and 72% respectively. In GMRES method applied to larger matrix orders ($n = 2048, 8192$) MCMCMI preconditioner shows a dramatic improvement of 94%. On the other hand, using the BiCGStab solver, MCMCMI-based preconditioner provides performance comparable with the reference solution for small matrices ($n =$

128, 512). For larger system matrices ($n = 2048, 8192$) MCMCMI preconditioner reduces the number of iterations by 26% and 27% respectively.

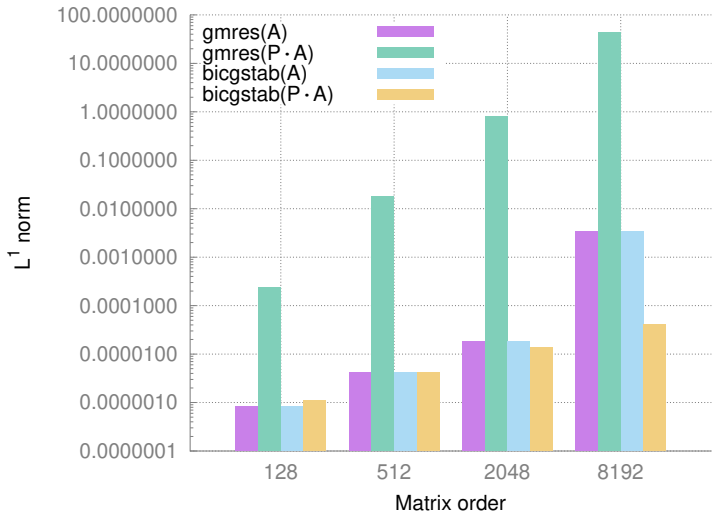

Figure 3: **Matrix order vs L^1 norm values in GMRES and BiCGStab, BOUT++.**

We compare the L^1 norm values for our approximate solutions from GMRES and BiCGStab for the BOUT++ test problems in Figure 3. The highest values of L^1 norm are provided by the preconditioned linear system solve in the GMRES method. L^1 norm values for non-preconditioned solutions in GMRES and BiCGStab are comparable. Finally, the smallest values of L^1 norm are obtained by the preconditioned solution with the BiCGStab. One exception is the L^1 norm values for the smallest test matrix ($n = 128$).

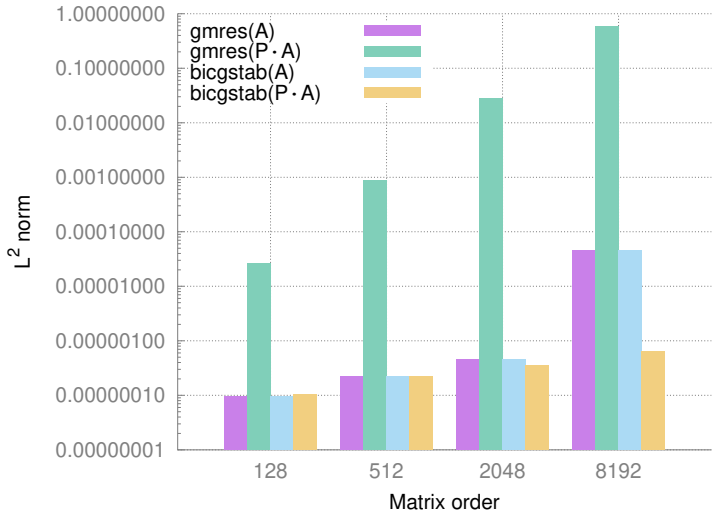

Figure 4: **Matrix order vs L^2 norm values in GMRES and BiCGStab, BOUT++.**

In Figure 4, we compare the L^2 norm values for our approximate solutions from GMRES and BiCGStab for the BOUT++ test problems. The L^2 norm values follow the same tendency as the L^1 norms described above (See Fig. 3 for details). The highest values are produced by the preconditioned solution with the GMRES method and the lowest—by the preconditioned solution with the BiCGStab algorithm. The L^∞ norm values are compared in Figure 5. L^∞ norms exhibit behaviour similar to the L^1 and L^2 norms. Overall, the L^∞ norms are the smallest amongst all norms. With rare exceptions the preconditioned system solutions resulted in the highest (GMRES) and the lowest (BiCGStab) norm values. We note that a left preconditioner is being used within the GMRES method and, hence, at each

iteration k, the preconditioned residual

$$||P^{-1}\left(b - Ax_k\right)||_2$$

is minimimised, where x_k is a member of the Krylov subspace defined by

$$\text{span}\left\{P^{-1}r_0, P^{-1}AP^{-1}r_0, \ldots, \left(P^{-1}A\right)^k P^{-1}r_0\right\}$$

with $r_0 = b - Ax_0$. Now,

$$\frac{||P^{-1}\left(b - Ax_k\right)||_2}{||P^{-1}||_2} \leq ||b - Ax_k||_2 \leq ||P||_2||P^{-1}\left(b - Ax_k\right)||_2$$

and, hence, left preconditioned GMRES is not minimising the L^2 norm of the residual and the values of $||P||_2$ and $||P^{-1}||_2$ are likely to be such that whilst $||P^{-1}\left(b - Ax_k\right)||_2$ can be small, the value of $||b - Ax_k||_2$ can be large. There is no minimisation property for the iterations of BiCGStab but these results lead to the question: would right preconditioned GMRES lead to similar results to BiCGStab? In the future, we would like to investigate this with larger problem sizes.

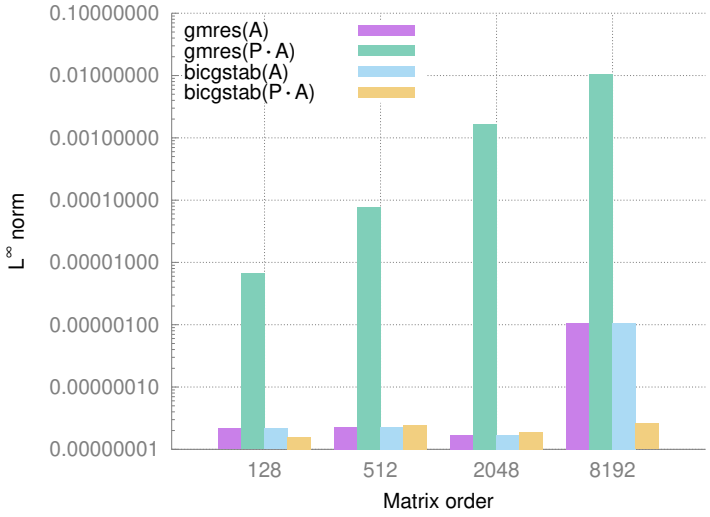

Figure 5: **Matrix order vs L^∞ norm values in GMRES and BiCGStab, BOUT++.**

In Table 1, we provide the details for the matrices that were extracted from Nektar++. We compare the number of GMRES and BiCGSTAB steps in Figure 6. Since the iterations are higher than non-preconditioned GMRES method, the preconditioner failed for the GMRES method for all except the Helmholtz problems. However, the preconditioner is successful for the BICGSTAB method. Higher L^2 and L^∞ values also supports the failure of the preconditioner for the GMRES method, likewise success for the BICGSTAB method as shown in the Figures 7 and 8.

In the future, we would like to investigate the use of the MCMCMI preconditioner with test problems that arise from anisotropic problems to see if the performance is markedly different.

Matrix name	Label	n	Mode — Order	Int timestep
Helmholtz	H	1312	2D modal	
Steady Adv-Diff	SAD	2281	2D modal	
Unsteady Adv-Diff 1	UAD1	225	Order 1	001
Unsteady Adv-Diff 2	UAD2	225	Order 1	0001
Unsteady Adv-Diff 3	UAD3	225	Order 2	001
Unsteady Adv-Diff 4	UAD4	225	Order 2	0001

Table 1: **Nektar++ matrices used in MCMCMI preconditioner experiments.**

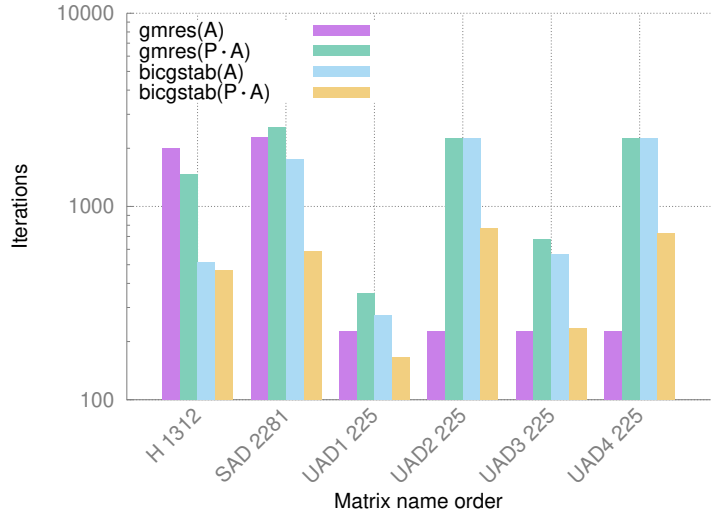

Figure 6: **Matrix order vs GMRES and BiCGStab iteration steps, Nektar++.**

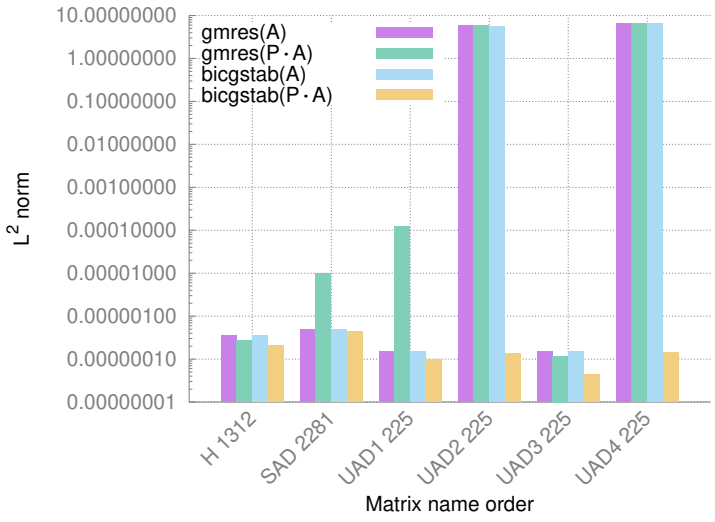

Figure 7: **Matrix order vs L^2 norm values in GMRES and BiCGStab, Nektar++.**

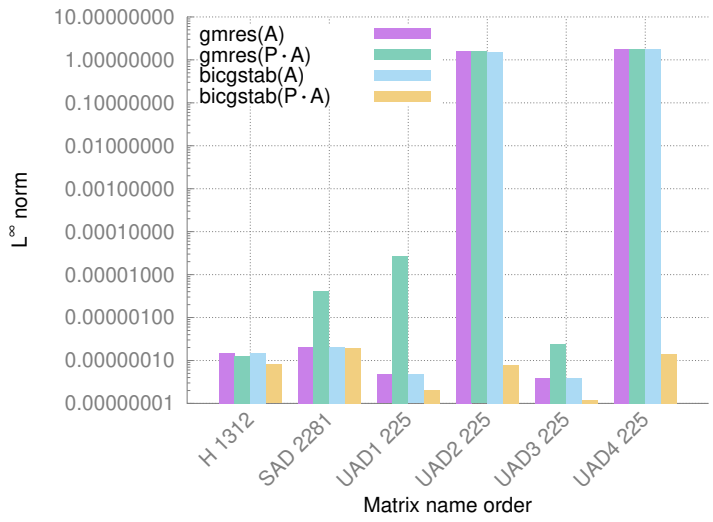

Figure 8: **Matrix order vs L^∞ norm values in GMRES and BiCGStab, Nektar++.**

4 Numerical results for operator-based preconditioners

Here, we focus on operator-based preconditioners. An understanding of the underlying mathematical operators and the numerical properties of discretised version can provide great insight into choice of preconditioner. For example, a standard finite-element discretisation of the Laplacian operator will, in general, have condition number that is inversely proportion to h^2 for 2D problems and h^3 for 3D, where h is the mesh size. Thus, halving h will increase the condition number of factors of 4 or 8, respectively. This can cause a dramatic increase in iterations if no preconditioner, or an ineffective preconditioner, is used.

Suppose we have a problem that couples together 3 different variables, then the matrix A will naturally split into a block 3×3 format:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \tag{2}$$

For such problems, it will be natural to use a block-diagonal preconditioner of the form:

$$P = \begin{bmatrix} P_{1,1} & 0 & 0 \\ 0 & P_{2,2} & 0 \\ 0 & 0 & P_{3,3} \end{bmatrix}, \tag{3}$$

where the dimension of each block directly tallies with the dimension of the associated block in (2). Alternatively, P could be block upper or lower triangular, or take a constraint preconditioner format:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & A_{1,3} \\ P_{2,1} & P_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}.$$

For symmetric A, Wathen [8] provides a good overview of block preconditioners and we note that factorizations of constraint preconditioners can be generated implicitly [3]. Constraint preconditioners have been extended for non-symmetric problems in [7] and, due to time restrictions, are not considered for the test case considered in this report.

Here, we consider the SD1D test case [4], which uses BOUT++ to simulate a plasma fluid in one dimension (along the magnetic field) that interacts with a neutral gas fluid. Unlike the nonlinear diffusion example, this test case provides a preconditioner as part of the model in an operator-based manner. SD1D has a number of different cases. For Case-03, the equations for the plasma density n, pressure p and momentum $m_i n V_{||i}$ are evolved:

$$\begin{aligned}
\frac{\partial n}{\partial t} &= -\nabla \cdot \left(\mathbf{b} V_{||} n\right) + S_n - S && (4)\\
\frac{\partial}{\partial t}\left(\frac{3}{2}p\right) &= -\nabla \cdot \mathbf{q} + V_{||}\partial_{||} p + S_p - E - R && (5)\\
\frac{\partial}{\partial t}\left(m_i n V_{||}\right) &= -\nabla \cdot \left(m_i n V_{||} \mathbf{b} V_{||}\right) - \partial_{||} p - F && (6)\\
j_{||} &= 0 \\
T_i &= T_e = \frac{1}{2}\frac{p}{en} \\
\mathbf{q} &= \frac{5}{2} p \mathbf{b} V_{||} - \kappa_{||e} \partial_{||} T_e
\end{aligned}$$

Which has a conserved energy:

$$\int_V \left[\frac{1}{2} m_i n V_{||i}^2 + \frac{3}{2} p\right] dV$$

The heat conduction coefficient $\kappa_{||e}$ is a nonlinear function of temperature T_e:

$$\kappa_{||e} = \kappa_0 T_e^{5/2}$$

where κ_0 is a constant. See [4] for further details. Operators are:

$$\partial_{||} f = \mathbf{b} \cdot \nabla f \qquad \nabla_{||} f = \nabla \cdot (\mathbf{b} f) \tag{7}$$

This nonlinear problem is solved by using CVODE from the SUNDIALS library [9], which uses a Newton method. At the heart of the simulation, a large number of systems of the form (2) are solved, where $A = I - \gamma J$ and J are Jacobian matrices for a computed value of γ : for clarity, we will assume that J has the same block structure as A and $J_{1,j}$ are derived via (4), $J_{2,j}$ are derived via (5), $J_{3,j}$ are derived via (6), $J_{j,1}$ are formed by taking the derivative with respect to n, $J_{j,2}$ are formed by taking the derivative with respect to p and $J_{j,3}$ are formed by taking the derivative with respect to $m_i n V_{||}$. We note that $J_{2,1}$ and $J_{2,2}$ contain terms that including $\partial^2_{||}$.

The preconditioner provided within SD1D takes the following block-diagonal, operator form:

$$P_0 = \begin{bmatrix} I & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial^2_{||} & 0 \\ 0 & 0 & I \end{bmatrix}. \tag{8}$$

We note that 5 shows that $\partial^2_{||}$ is applied to $\frac{1}{2} T_e$ and not p, and hence, we propose the following block-diagonal, operator-form preconditioner:

$$P_1 = \begin{bmatrix} I & 0 & 0 \\ 0 & (I - \gamma \frac{1}{3n} \partial^2_{||}) n & 0 \\ 0 & 0 & I \end{bmatrix}. \tag{9}$$

We note that application of the preconditioner P_1 will be more expensive than P_0. Finally, for Case-03, we also try a block lower triangular preconditioner that additionally incorporates the $\partial_{||}$ from (6):

$$P_2 = \begin{bmatrix} I & 0 & 0 \\ 0 & (I - \gamma \frac{1}{3n} \partial^2_{||}) n & 0 \\ 0 & -\gamma \partial_{||} & I \end{bmatrix}. \tag{10}$$

Case-04 from SD1D additionally couples the above plasma equations to a similar set of equations for the neutral gas density, pressure, and parallel momentum. A fixed particle and power source is used here, and a 20% recycling fraction. Exchange of particles, momentum and energy between neutrals and plasma occurs through ionisation, recombination and charge exchange. If we sub-divide A according to the variables that are being evolved, we now have a block 6×6 structure. The provided preconditioner P_0 is now

$$P_0 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial^2_{||} & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma \frac{2}{3} \partial^2_{||} & \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}. \tag{11}$$

Using the properties of the neutral, we instead propose

$$P_1 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial^2_{||} & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma D_n \frac{2}{3} \partial^2_{||} & \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}, \tag{12}$$

where D_n is a diffusion term. As for case-03, we also propose a block lower-triangular preconditioner that incorporates the $\partial_{||}$ from (6) and the $\partial_{||}$ from the corresponding equation for neutral momentum:

$$P_2 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial^2_{||} & 0 & 0 & 0 & 0 \\ 0 & -\gamma \partial_{||} & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma D_n \frac{2}{3} \partial^2_{||} & \\ 0 & 0 & 0 & 0 & -\gamma \partial_{||} & I \end{bmatrix}. \tag{13}$$

Finally, we consider a preconditioner that is identical to P_2 but the $\partial_{||}$ relating to the neutral momentum equation is dropped:

$$P_3 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma\frac{2}{3}\partial_{||}^2 & 0 & 0 & 0 & 0 \\ 0 & -\gamma\partial_{||} & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma D_n \frac{2}{3}\partial_{||}^2 & \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}. \tag{14}$$

In Figure 9, we compare the number of times the time derivatives (right-hand sides) were computed for Case-03 and Case-04 for the different preconditioners. We also compare what happens if no preconditioner is used for the smaller problems. The smaller matrix orders are from the default problem set-up (i.e., 200 mesh points for each variable). For Case-03, we also compare the preconditioners for the discretization with 1000 mesh points per variable and, for Case-04, 400 mesh points per variable. If we consider the smaller version of Case-03, all of the preconditioners reduce the number of right-hand side evaluations by roughly a factor of 36 with P_2 producing the lowest number of evaluations; for the larger problem, we observe that P_2 also has the lowest number of evaluations with an 18% reduction compared to P_0. In Figure 10, we compare the wall clock time to run the simulations. Just one MPI process was used with one OpenMP thread because of the relatively small problem sizes. For the larger version of Case-03, we see a 10% improvement, with respect to wall-clock time, when using the block lower-triangular preconditioner compared to the original preconditioner. These savings are relatively modest because the density n remains close to uniform throughout the simulation.

For Case-04, we observe that all of the preconditioners substantially decrease the number of evaluations and the wall clock time compared to using no preconditioner. For both problem sizes, compared to the original preconditioner, preconditioners P_1, P_2 and P_3 are reducing the number of evaluations and wall clock time by a third. Thus, for the expert user, it is possible to incorporate more sophisticated operator-based preconditioners within BOUT++.

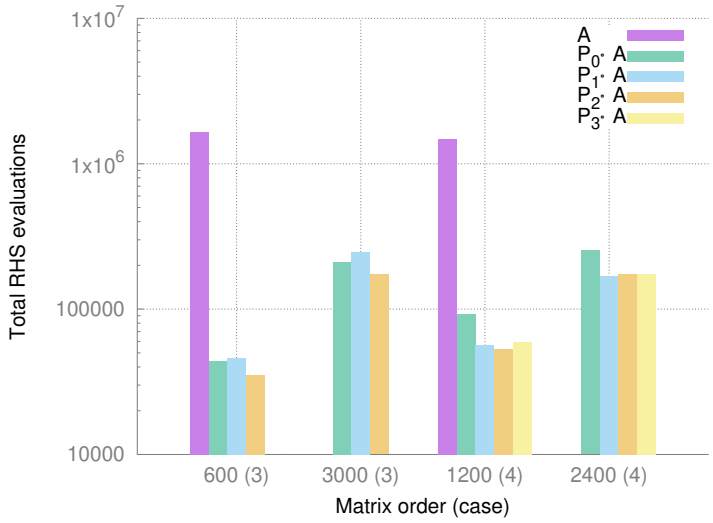

Figure 9: **Total number of right-hand side evaluations performed during the whole simulation for Case-03 and Case-04 with different preconditioners.** For Case-03, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided. For Case-04, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided.

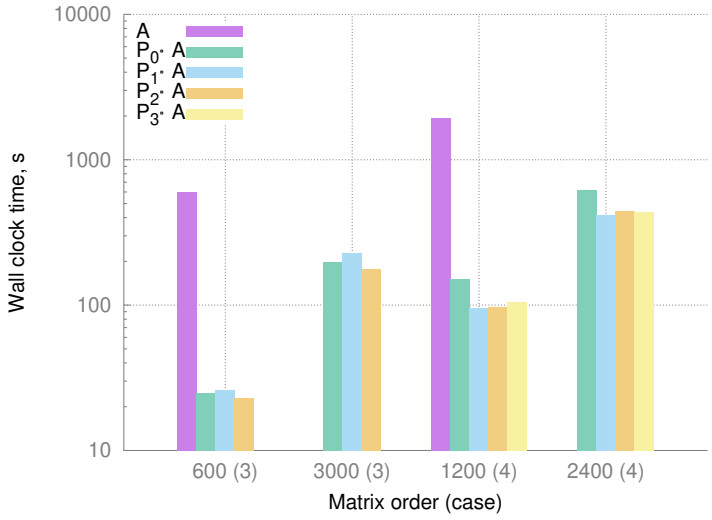

Figure 10: **Total wall clock time (seconds) for the whole simulation for Case-03 and Case-04 with different preconditioners.** For Case-03, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided. For Case-04, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided.

5 Proposed roadmap for including new preconditioners within BOUT++ and Nektar++

Prior to including custom new preconditioners into BOUT++ and Nektar++ [2] it would be the best to have the exact testing scenarios of interest to UKAEA ready. Once the testing scenarios are available and working at scale close to the real problems UKAEA want to solve, it would be the right time to investigate preconditioner deployment. Ideally the new preconditioners need to be reformulated in a matrix-free manner. Once these formulations are available they could be integrated into PETSc, SUNDIALS and other solver packages powering BOUT++ and Nektar++. This way it would bring more benefit to the user community. The user base of PETSc and SUNDIALS is likely to be much larger than that of BOUT++ and Nektar++. This would also ensure that only minimal changes would be required to BOUT++ and Nektar++ in order to leverage computational advantages of new preconditioners.

In summary the following steps are required for a successful deployment of new preconditioners into BOUT++ and Nektar++:

1. Design testing scenarios in native BOUT++ and Nektar++. These scenarios should reflect real-world problems UKAEA solves at the moment. Pay attention to scale at which simulations should be performed.

2. Identify (matrix-free) methods and underlying solver packages (PVODE, CVODE, PETSc, SUNDIALS) that BOUT++ and Nektar++ employ to solve those problems.

3. Reformulate MCMCMI-based preconditioner in the operator (matrix-free) form.

4. Implement MCMCMI preconditioner in operator form and test it thoroughly.

5. Integrate operator-based MCMCMI preconditioner into solver packages identified in Step 2.

6. Solve testing scenarios identified in Step 1 using standard (native) BOUT++ and Nektar++ methods as well as new MCMCMI preconditioners.

7. Compare computational performance of native vs MCMCMI-based preconditioners.

6 Conclusion

Including MCMCMI-based preconditioners into either BOUT++ or Nektar++ simulation software is difficult at the moment. First, the MCMCMI code needs to be reformulated in the matrix-free manner. Then it can be integrated into solver packages (PETSc, SUNDIALS) powering solving abilities of BOUT++ and Nektar++. Reformulation and implementation of MCMCMI in the operator form will take 3–6 months. Integration of operator-based MCMCMI into PETSc and SUNDIALS will take another 3–6 months. *Therefore, to obtain preliminary results of MCMCMI performance over the test problems the STFC team decided to extract the system matrices from the relevant BOUT++ and Nektar++ testing scenarios.* In the process the team discovered the (i) it was not possible to extract the matrices for some scenarios due to their inherent matrix-free nature, (ii) some testing scenarios of interest to UKAEA are not available yet in either BOUT++ or Nektar++. These need to be designed and developed further.

Acknowledgements

The team would like to thank Dr. Benjamin Dudson from the University of York and Dr. Chris Cantwell from the Imperial College London for their time, help and guidance in technical aspects of BOUT++ and Nektar++ functionality.

References

[1] V. Alexandrov, A. Lebedev, E. Sahin, and S. Thorne. Linear systems of equations and preconditioners relating to the NEPTUNE Programme: a brief overview. Technical Report 2047353-TN-02, UKAEA, 2021.

[2] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. D. Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer physics communications*, 192:205–219, 2015.

[3] H. S. Dollar, N. I. Gould, W. H. Schilders, and A. J. Wathen. Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems. *SIAM Journal on Matrix Analysis and Applications*, 28(1):170–189, 2006.

[4] B. Dudson. SD1D. online repository, 2016. URL `https://github.com/boutproject/SD1D`.

[5] B. Dudson, P. Hill, and J. Parker. BOUT++. online repository, 2020. URL `http://boutproject.github.io`.

[6] S. Sherwin, M. Kirby, C. Cantwell, and D. Moxey. Nektar++. online, 2021. URL `https://www.nektar.info`.

[7] S. Thorne. Implicit-factorization preconditioners for non-symmetric problems. Technical Report 2047353-TN-04, UKAEA, 2021.

[8] A. Wathen. Preconditioning. *Acta Numerica*, 24:329 – 376, 2015.

[9] C. S. Woodward, D. R. Reynolds, A. C. Hindmarsh, D. J. Gardner, and C. J. Balos. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. online, 2021. URL `https://computing.llnl.gov/projects/sundials`.

Science and Technology Facilities Council

Implicit-factorization preconditioners for non-symmetric problems

Technical Report 2047353-TN-04

Maksims Abaļenkovs* Vassil Alexandrov* Anton Lebedev* Emre Sahin*
Sue Thorne**

July 2021

1 Introduction

In this report, we extend the class of constraint preconditioners from symmetric problems to non-symmetric problems. We consider the theoretical properties and demonstrate their effectiveness on a set of test problems inspired by the Hasegawa-Wakatani problem.

2 Constraint-style preconditioners

Let us assume that

$$\mathscr{A} = \begin{pmatrix} H & C \\ B & -D \end{pmatrix}, \tag{1}$$

where $H \in \mathbb{R}^{n\times n}$, $B, C^T \in \mathbb{R}^{m\times n}$ and $D \in \mathbb{R}^{m\times m}$ subject to $m \leq n$. We always assume that $\mathscr{A}$ is non-singular. We consider the use of a preconditioner of the form

$$\mathscr{P} = \begin{pmatrix} G & C \\ B & -D \end{pmatrix}, \tag{2}$$

where $G \in \mathbb{R}^{n\times n}$.

2.1 Constraint-style preconditioners: symmetric case

The case when $D = 0$, $B = C^T$ and $H = H^T$ was analysed by Keller, Gould and Wathen [6]:

Theorem 2.1. *Let $\mathscr{A} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric and indefinite matrix of the form*

$$\mathscr{A} = \begin{pmatrix} H & B^T \\ B & 0 \end{pmatrix},$$

where $H \in \mathbb{R}^{n\times n}$ is symmetric and $B \in \mathbb{R}^{m\times n}$ is of full rank. Assume Z is an $n \times (n-m)$ basis for the nullspace of B. Preconditioning $\mathscr{A}$ by a matrix of the form

$$\mathscr{P} = \begin{pmatrix} G & B^T \\ B & 0 \end{pmatrix},$$

where $G \in \mathbb{R}^{n\times n}$ is symmetric, and $B \in \mathbb{R}^{m\times n}$ is as above, implies that the matrix $\mathscr{P}^{-1}\mathscr{A}$ has

*The authors are with the Hartree Centre, STFC Daresbury Laboratory, Sci-Tech Daresbury, Keckwick, Daresbury, Warrington, WA4 4AD, UK.

**Sue Thorne is with the Hartree Centre, STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK. Email contact: sue.thorne@stfc.ac.uk

1. *an eigenvalue at 1 with multiplicity* $2m$;
2. $n-m$ *eigenvalues* λ *which are defined by the generalized eigenvalue problem*

$$Z^T H Z x_z = \lambda Z^T G Z x_z. \tag{3}$$

This accounts for all of the eigenvalues.

Assume, in addition, that $Z^T GZ$ *is positive definite. Then* $\mathscr{P}^{-1}\mathscr{A}$ *has the following* $m+i+j$ *linearly independent eigenvectors:*

1. m *eigenvectors of the form* $\left[0^T, y^T\right]^T$ *corresponding to the eigenvalue 1 of* $\mathscr{P}^{-1}\mathscr{A}$;
2. i $(0 \le i \le n)$ *eigenvectors of the form* $\left[w^T, y^T\right]^T$ *corresponding to the eigenvalue 1 of* $\mathscr{P}^{-1}\mathscr{A}$, *where the components* w *arise from the generalized eigenvalue problem* $Hw = Gw$;
3. j $\left(0 \le j \le n-m\right)$ *eigenvectors of the form* $\left[x_z^T, 0^T, y^T\right]^T$ *corresponding to the eigenvalues of* $\mathscr{P}^{-1}\mathscr{A}$ *note equal to 1, where the components* x_z *arise from the generalized eigenvalue problem* $Z^T HZ x_z = \lambda Z^T GZ x_z$ *with* $\lambda \neq 1$.

The case when when $B = C^T$, $H = H^T$ and D symmetric and positive definite has been analysed by a number of different authors [2, 3, 4] and can be summarised in the following theorems:

Theorem 2.2. *Let* $\mathscr{A} \in \mathbb{R}^{(n+m)\times(n+m)}$ *be a symmetric and indefinite matrix of the form*

$$\mathscr{A} = \begin{pmatrix} H & B^T \\ B & -D \end{pmatrix},$$

where $H \in \mathbb{R}^{n\times n}$ *is symmetric,* $B \in \mathbb{R}^{m\times n}$ *is of full rank and* $D \in \mathbb{R}^{m\times m}$ *is symmetric and positive definite. Preconditioning* $\mathscr{A}$ *by a matrix of the form*

$$\mathscr{P} = \begin{pmatrix} G & B^T \\ B & -D \end{pmatrix},$$

where $G \in \mathbb{R}^{n\times n}$ *is symmetric, and* $B \in \mathbb{R}^{m\times n}$ *and* $D \in \mathbb{R}^{m\times m}$ *are as above, implies that the matrix* $\mathscr{P}^{-1}\mathscr{A}$ *has*

1. *an eigenvalue at 1 with multiplicity* m;
2. n *eigenvalues* λ *which are defined by the generalized eigenvalue problem*

$$\left(H + B^T D^{-1} B\right) x = \lambda \left(G + B^T D^{-1} B\right) x. \tag{4}$$

This accounts for all of the eigenvalues.

Dollar *et al.* [4] have extended Theorem 2.2 to the case when D is symmetric and positive semi-definite:

Theorem 2.3. *Let* $\mathscr{A} \in \mathbb{R}^{(n+m)\times(n+m)}$ *be a symmetric and indefinite matrix of the form*

$$\mathscr{A} = \begin{pmatrix} H & B^T \\ B & -D \end{pmatrix},$$

where $H \in \mathbb{R}^{n\times n}$ *is symmetric,* $B \in \mathbb{R}^{m\times n}$ *is of full rank and* $D \in \mathbb{R}^{m\times m}$ *is symmetric and positive semi-definite with rank* l, *where* $0 < l < m$. *Assume that* D *is factored as* $D = ESE^T$, *where* $E \in \mathbb{R}^{m\times l}$ *and* $S \in \mathbb{R}^{l\times l}$ *is nonsingular,* $F \in \mathbb{R}^{m\times(m-l)}$ *is a vasis for the nullspace of* E^T *and* $\begin{bmatrix} E & F \end{bmatrix}$ *is orthogonal. Let the columns of* $N \in \mathbb{R}^{n\times(n-m+l)}$ *span the nullspace of* $F^T B$. *Preconditioning* $\mathscr{A}$ *by a matrix of the form*

$$\mathscr{P} = \begin{pmatrix} G & B^T \\ B & -D \end{pmatrix},$$

where $G \in \mathbb{R}^{n\times n}$ *is symmetric, and* $B \in \mathbb{R}^{m\times n}$ *and* $D \in \mathbb{R}^{m\times m}$ *are as above, implies that the matrix* $\mathscr{P}^{-1}\mathscr{A}$ *has*

1. *an eigenvalue at 1 with multiplicity* $2m-l$;
2. $n-m+l$ *eigenvalues* λ *which are defined by the generalized eigenvalue problem*

$$N^T\left(H + B^T ES^{-1}E^T B\right) Nz = \lambda N^T\left(G + B^T ES^{-1}E^T B\right) Nz. \tag{5}$$

This accounts for all of the eigenvalues.

2.2 Constraint-style preconditioners: nonsymmetric case

We will now extend Theorems 2.1 and 2.2 to the non-symmetric case.

2.3 D non-singular

Theorem 2.4. *Let $\mathscr{A} \in \mathbb{R}^{(n+m)\times(n+m)}$, $m \le n$, be a matrix of the form*

$$\mathscr{A} = \begin{pmatrix} H & C \\ B & -D \end{pmatrix},$$

where $H \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{n\times m}$ are of full rank and $D \in \mathbb{R}^{m\times m}$ is non-singular. Preconditioning $\mathscr{A}$ by a matrix of the form

$$\mathscr{P} = \begin{pmatrix} G & C \\ B & -D \end{pmatrix},$$

where $G \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{n\times m}$ and $D \in \mathbb{R}^{m\times m}$ are as above, implies that the matrix $\mathscr{P}^{-1}\mathscr{A}$ has

1. *an eigenvalue at 1 with multiplicity m;*
2. *n eigenvalues λ which are defined by the generalized eigenvalue problem*

$$\left(H + CD^{-1}B\right)x = \lambda\left(G + CD^{-1}B\right)x. \tag{6}$$

This accounts for all of the eigenvalues.

Proof. The eigenvalues of $\mathscr{P}^{-1}\mathscr{A}$ may be derived by considering the generalized eigenvalue problem

$$\begin{pmatrix} H & C \\ B & -D \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \lambda\begin{pmatrix} G & C \\ B & -D \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \tag{7}$$

Premultiplying (7) by the non-singular matrix

$$\begin{pmatrix} I & CD^{-1} \\ 0 & -D^{-1} \end{pmatrix}$$

gives the equivalent generalized eigenvalue problem

$$\begin{pmatrix} H + CD^{-1}B & 0 \\ -D^{-1}B & I \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \lambda\begin{pmatrix} G + CD^{-1}B & 0 \\ -D^{-1}B & I \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

Thus, there are m eigenvalues equal to 1 and the remaining n eigenvalues are defined by the generalized eigenvalue problem

$$\left(H + CD^{-1}B\right)x = \lambda\left(G + CD^{-1}B\right)x. \tag{8}$$

□

2.4 $D = 0$

Theorem 2.5. *Let $\mathscr{A} \in \mathbb{R}^{(n+m)\times(n+m)}$, $m \le n$, be a matrix of the form*

$$\mathscr{A} = \begin{pmatrix} H & C \\ B & 0 \end{pmatrix},$$

where $H \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{n\times m}$ are of full rank. Let the columns of $Z_B \in \mathbb{R}^{n\times(n-m)}$ span the nullspace of B and the columns of $Z_C \in \mathbb{R}^{n\times(n-m)}$ span the nullspace of C^T. Preconditioning $\mathscr{A}$ by a matrix of the form

$$\mathscr{P} = \begin{pmatrix} G & C \\ B & 0 \end{pmatrix},$$

where $G \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{n\times m}$ are as above, implies that the matrix $\mathscr{P}^{-1}\mathscr{A}$ has

1. *$2m$ eigenvalues of equal to 1;*

2. *the remaining $n-m$ eigenvalues, λ, are defined by the generalized eigenvalue problem*

$$Z_C^T H Z_B x_z = \lambda Z_C^T G Z_B x_z. \tag{9}$$

This accounts for all of the eigenvalues.

Proof. The eigenvalues of $\mathscr{P}^{-1}\mathscr{A}$ may be derived by considering the generalized eigenvalue problem

$$\begin{pmatrix} H & C \\ B & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} G & C \\ B & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}, \tag{10}$$

where $\lambda \in \mathbb{C}$, $\lambda \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}^m$. Let

$$B = U_B \begin{pmatrix} \Sigma_B & 0 \end{pmatrix}\begin{pmatrix} Y_B^T \\ Z_B^T \end{pmatrix}, \quad C^T = U_C \begin{pmatrix} \Sigma_C & 0 \end{pmatrix}\begin{pmatrix} Y_C^T \\ Z_C^T \end{pmatrix}$$

be the singular-value decompositions of B and C with $Y_B, Y_C \in \mathbb{R}^{n\times m}$. Note that $Z_B, Z_C \in \mathbb{R}^{n\times(n-m)}$ span the nullspace of B and C^T, respectively.

If we substitute in $x = Y_B x_Y + Z_B x_Z$ into (10) and premultiply the equation by the nonsingular matrix

$$\begin{pmatrix} Y_C^T & 0 \\ Z_C^T & 0 \\ 0 & I \end{pmatrix},$$

where Y_B and Y_C are n by m matrices whose columns span the range space of B^T and C, respectively, then we obtain

$$\begin{pmatrix} Y_C^T H Y_B & Y_C^T H Z_B & Y_C^T C \\ Z_C^T H Y_B & Z_C^T H Z_B & 0 \\ B Y_B & 0 & 0 \end{pmatrix}\begin{pmatrix} x_Y \\ x_Z \\ y \end{pmatrix} = \lambda \underbrace{\begin{pmatrix} Y_C^T G Y_B & Y_C^T G Z_B & Y_C^T C \\ Z_C^T G Y_B & Z_C^T G Z_B & 0 \\ B Y_B & 0 & 0 \end{pmatrix}}_{\tilde{\mathscr{P}}}\begin{pmatrix} x_Y \\ x_Z \\ y \end{pmatrix}. \tag{11}$$

If we pre-multiply (11) by $\tilde{\mathscr{P}}^{-1}$, then we obtain an equivalent eigenvalue problem of the form

$$\begin{pmatrix} I & 0 & 0 \\ \Theta_1 & (Z_C^T G X_B)^{-1} Z_C^T H Z_B & 0 \\ \Theta_2 & \Theta_3 & I \end{pmatrix}\begin{pmatrix} x_Y \\ x_Z \\ y \end{pmatrix} = \lambda \begin{pmatrix} x_Y \\ x_Z \\ y \end{pmatrix}, \tag{12}$$

where the exact definition of Θ_1, Θ_2 and Θ_3 are not important for the proof. Hence, $\mathscr{P}^{-1}\mathscr{A}$ has $2m$ eigenvalues equal to 1 and the remaining eigenvalues are defined by the eigenvalue problem generalized eigenvalue problem (9).

□

We note that when $\mathscr{A}$ and $\mathscr{P}$ are no-longer symmetric, some of the non-unitary eigenvalues may be complex.

3 Implicit-factorization constraint preconditioners

In [4], the authors derive a number of factorizations for generating symmetric constraint preconditioners. In the following, we will assume that the rows and columns of H have been ordered in such a manner that we can partition $B \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{n\times m}$, $G \in \mathbb{R}^{n\times n}$ and $H \in \mathbb{R}^{n\times n}$ as

$$B = \begin{pmatrix} B_1 & B_2 \end{pmatrix}, \tag{13}$$

$$C = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}, \tag{14}$$

$$G = \begin{pmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{pmatrix}, \tag{15}$$

$$H = \begin{pmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{pmatrix}, \tag{16}$$

where $B_1 \in \mathbb{R}^{m\times m}$ and $C_1 \in \mathbb{R}^{m\times m}$ are non-singular, $G_{1,1} \in \mathbb{R}^{m\times m}$ and $H_{1,1} \in \mathbb{R}^{m\times m}$. For coupled multi-physics problems, this ordering is implicitly available through the nature of the problems. As in [4], we form factors of the form

$$
\begin{aligned}
L &= \begin{pmatrix} L_{1,1} & L_{1,2} & L_{1,3} \\ L_{2,1} & L_{2,2} & L_{2,3} \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix}, \\
N &= \begin{pmatrix} N_{1,1} & N_{1,2} & N_{1,3} \\ N_{2,1} & N_{2,2} & N_{2,3} \\ N_{3,1} & N_{3,2} & N_{3,3} \end{pmatrix}, \\
R &= \begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} \\ R_{2,1} & R_{2,2} & R_{2,3} \\ R_{3,1} & R_{3,2} & R_{3,3} \end{pmatrix}
\end{aligned}
$$

set some of the sub-blocks to zero whilst assuming other sub-blocks are invertible and relatively easy to solve with, and the sub-blocks are such that the product LNR forms a non-symmetric constraint preconditioner of the form

$$
\mathscr{P} = \begin{pmatrix} G & C \\ B & -D \end{pmatrix}.
$$

Without loss of generality, we fix $L_{1,3}$, $L_{2,2}$, $L_{2,3}$, $R_{2,2}$, $R_{3,1}$ and $R_{3,2}$ to be non-zero with $L_{2,2}$ and $R_{2,2}$ both non-singular. We use a Matlab script (see Appendix A) to generate all 62 possible implicit-factorization constraint preconditioners. We note that if $B = C^T$, $R_{3,1} = B_1$, $R_{3,2} = B_2$, $L_{1,3} = B_1^T$ and $L_{2,3} = B_2^T$, then we obtain the families given in [4].

Some of the non-symmetric implicit factorizations are only suitable for the case $D = 0$, for example

$$
\begin{aligned}
L &= \begin{pmatrix} L_{1,1} & 0 & L_{1,3} \\ L_{2,1} & L_{2,2} & L_{2,3} \\ L_{3,1} & 0 & 0 \end{pmatrix}, \\
N &= \begin{pmatrix} 0 & 0 & N_{1,3} \\ 0 & N_{2,2} & N_{2,3} \\ N_{3,1} & N_{3,2} & N_{3,3} \end{pmatrix}, \\
R &= \begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} \\ 0 & R_{2,2} & 0 \\ R_{3,1} & R_{3,2} & 0 \end{pmatrix}
\end{aligned}
$$

subject to

$$
\begin{aligned}
L_{3,1}N_{1,3}R_{3,1} &= B_1, \\
B_1R_{3,1}^{-1}R_{3,2} &= B_2, \\
L_{1,3}N_{3,1}R_{1,3} &= C_1, \\
L_{2,3}L_{1,3}^{-1}C_1 &= C_2
\end{aligned}
$$

produces

$$
\begin{aligned}
G_{1,1} &= L_{1,1}N_{1,3}R_{3,1} + L_{1,3}N_{3,3}R_{3,1} + L_{1,3}N_{3,1}R_{1,1}, \\
G_{1,2} &= L_{1,1}L_{3,1}^{-1}B_2 + L_{1,3}N_{3,3}R_{3,1}B_1^{-1}B_2 + C_1R_{1,3}^{-1}R_{1,2} + L_{1,3}N_{3,2}R_{2,2}, \\
G_{2,1} &= L_{2,1}L_{3,1}^{-1}B_1 + L_{2,2}N_{2,3}R_{3,1} + C_2C_1^{-1}L_{1,3}N_{3,3}R_{3,1} + C_2R_{1,3}^{-1}R_{1,1}, \\
G_{2,2} &= L_{2,2}N_{2,2}R_{2,2} + C_2C_1^{-1}L_{1,3}N_{3,2}R_{2,2} + L_{2,1}L_{3,1}^{-1}B_2 + C_2C_1^{-1}L_{1,3}N_{3,3}R_{3,1}B_1^{-1}B_2 \\
&\quad + L_{2,2}N_{2,3}R_{3,1}B_1^{-1}B_2.
\end{aligned}
$$

There are also some that are only suitable for non-singular D, for example,

$$\begin{aligned} L &= \begin{pmatrix} L_{1,1} & 0 & L_{1,3} \\ L_{2,1} & L_{2,2} & L_{2,3} \\ L_{3,1} & 0 & 0 \end{pmatrix}, \\ N &= \begin{pmatrix} 0 & 0 & N_{1,3} \\ 0 & N_{2,2} & N_{2,3} \\ N_{3,1} & N_{3,2} & N_{3,3} \end{pmatrix}, \\ R &= \begin{pmatrix} R_{1,1} & R_{1,2} & 0 \\ 0 & R_{2,2} & 0 \\ R_{3,1} & R_{3,2} & R_{3,3} \end{pmatrix} \end{aligned}$$

subject to

$$\begin{aligned} L_{3,1}N_{1,3}R_{3,1} &= B_1, \\ B_1 R_{3,1}^{-1} R_{3,2} &= B_2, \\ L_{3,1}N_{1,3}R_{33} &= D, \\ -\left(L_{1,1}N_{1,3} + L_{1,3}N_{3,3}\right)R_{3,1} &= C_1 D^{-1} B_1, \\ -\left(L_{2,1}N_{1,3} + L_{2,3}N_{3,3} + L_{2,2}N_{2,3}\right)R_{3,1}R_{3,1} &= C_1 D^{-1} B_1 \end{aligned}$$

produces

$$\begin{aligned} G_{1,1} &= -C_1 D^{-1} B_1 + L_{1,3}N_{3,1}R_{1,1}, \\ G_{1,2} &= -C_1 D^{-1} B_2 + \left(G_{1,1} + C_1 D^{-1} B_1\right) R_{1,1}^{-1} R_{1,2} + L_{1,3}N_{3,2}R_{2,2}, \\ G_{2,1} &= -C_2 D^{-1} B_1 + L_{2,3}N_{3,1}R_{1,1}, \\ G_{2,2} &= L_{2,2}N_{2,2}R_{2,2} + L_{2,3}N_{3,2}R_{2,2} + C_2 C^{-1} B_2 + L_{2,3}N_{3,1}R_{1,2}. \end{aligned}$$

4 Numerical tests

We will consider a test problem inspired by the 2D problem known as the Hasegawa-Wakatani problem, which is similar to incompressible fluid dynamics:

$$\begin{aligned} \frac{\partial n}{\partial t} &= -\{\phi, n\} + \alpha(\phi - n) - \kappa \frac{\partial \phi}{\partial z} + D_n \nabla_\perp^2 n \\ \frac{\partial \omega}{\partial t} &= -\{\omega, n\} + \alpha(\omega - n) + D_\omega \nabla_\perp^2 \omega \\ \nabla^2 \phi &= \omega\,. \end{aligned}$$

Here n is the plasma number density, $\omega := \vec{b}_0 \cdot \nabla \times \vec{v}$ is the vorticity with $\vec{v}$ being the $\vec{E} \times \vec{B}$ drift velocity in a constant magnetic field and $\vec{b}_0$ is the unit vector in the direction of the equilibrium magnetic field. The operator $\{\cdot, \cdot\}$ is the Poisson bracket.

The discretized version of the problem is described in [1] but we will consider a split implicit-explicit method where the Jacobian that needs solving at each Newton iteration is of the following form:

$$J = \begin{pmatrix} A & 0 & B \\ 0 & C & E \\ -M & K & 0 \end{pmatrix}, \tag{17}$$

where the constituent matrices are the following

$$\begin{aligned} A &= M + \Delta t\,(-D_\omega K)\,, \\ B &= \alpha \Delta t M, \\ C &= \Delta t\,(-\alpha M)\,, \\ E &= M + \Delta t\,(\alpha M - D_n K)\,. \end{aligned}$$

Here K and M are the stiffness and mass matrices, respectively. Note that we have permuted the rows and columns so the matrix will not directly map to that given in [1]. We tried to use BOUT++[5] directly to solve the Hasegawa-Wakatani problem and test our preconditioners but using PETSc with a constraint and preconditioner resulted in runtime errors, which might be due to the manner that PETSc was installed on the Hartree Centre's Scafell Pike cluster. Instead, we took advantage of the situation and created mass and stiffness matrices that use a finite-element discretization instead of finite difference. We used the same values of constants as used within the BOUT++ implementation and set Δt to be equal to the inverse of the number of rows in M.

We will compare the following preconditioning strategies:

- A block-diagonal preconditioner
$$P_D = \begin{pmatrix} A & 0 & 0 \\ 0 & C & 0 \\ 0 & 0 & I \end{pmatrix};$$
- A constraint preconditioner with $G = I$
$$P_1 = \begin{pmatrix} I & 0 & B \\ 0 & I & E \\ -M & K & 0 \end{pmatrix};$$
- A constraint preconditioner with $G_{2,2} = I$ and the remainder of G zero:
$$P_2 = \begin{pmatrix} 0 & 0 & B \\ 0 & I & E \\ -M & K & 0 \end{pmatrix};$$
- A constraint preconditioner with $G_{2,2} = C$ and the remainder of G zero:
$$P_3 = \begin{pmatrix} 0 & 0 & B \\ 0 & C & E \\ -M & K & 0 \end{pmatrix};$$
- An implicit-factorization constraint preconditioner with:
$$\begin{aligned} L &= \begin{pmatrix} -I & 0 & I \\ -\frac{D_\omega D_n \Delta t}{\alpha} K M^{-1} K M^{-1} & I & \frac{1}{\alpha\Delta t}\left((1+\alpha\gamma)M - \gamma D_n K\right) M^{-1} \\ I & 0 & 0 \end{pmatrix}, \\ N &= \begin{pmatrix} 0 & 0 & -M \\ 0 & -\frac{1}{\alpha\Delta t}\left((1+\alpha\gamma)M - \gamma D_n K\right) M^{-1} K & \frac{D_\omega(1+\alpha\Delta t)}{\alpha} K \\ \alpha\Delta t M & K & -\gamma D_\omega K \end{pmatrix}, \\ R &= \begin{pmatrix} 0 & -\frac{D_\omega}{\alpha} M^{-1} K M^{-1} K & I \\ 0 & I & 0 \\ I & -M^{-1} K & 0 \end{pmatrix}. \end{aligned}$$

Note that, with the exception of preconditioner P_1, we do not explicitly form the preconditioner and we instead apply them by exploiting the block structures. In Tables 1 and 2, we report the number of iterations to reduce the relative residual by a factor of 10^{-6} and the times for solving our test problems using Matlab's GMRES function with no restarting. Note that the preconditioners have not been optimized with respect to time so these values are only indicative. Preconditioners P_1 and P_5 produce the best iteration counts but we note that for larger problems, factoring P_1 via a direct method will become extremely expensive. Additionally, alternative choices for the blocks in the implicit factorization preconditioner may increase the number of iterations but make the preconditioner much cheaper to apply. For example, solves with the mass matrix can be well-approximated using the Chebyshev semi-iteration [7] and solves involving the stiffness matrix may be approximated with a multigrid method: this was very successfully done within the symmetric constraint preconditioner context for PDE-constrained problems [8].

In Tables 1 and 2, we report the number of iterations to reduce the relative residual by a factor of 10^{-6} and the times for solving our test problems using Matlab's GMRES function with restarting set to 10. Here, preconditioner P_2 failed to converge but we see similar results to non-restarted GMRES for preconditioners P_1 and P_5. Note that by using the restarted version of GMRES, we were able to solve larger problems.

n	m	P_D	P_1	P_2	P_3	P_4
450	225	57	3	43	61	2
1922	961	103	2	86	122	2
7938	3969	192	2	172	239	2

Table 1: Number of preconditioned GMRES iterations to reduce the relative residual by a factor of 10^{-6}.

n	m	P_D	P_1	P_2	P_3	P_4
450	225	0.029	0.020	0.037	0.051	0.015
1922	961	0.21	0.077	0.17	0.31	0.27
7938	3969	2.18	0.37	0.37	3.97	8.63

Table 2: Time (in seconds) for preconditioned GMRES to reduce the relative residual by a factor of 10^{-6}.

n	m	P_D	P_1	P_2	P_3	P_4
450	225	404	3	-	15	2
1922	961	725	2	-	649	2
7938	3969	2078	2	-	2255	2
32258	16129	4514	2	-	7875	2

Table 3: Number of preconditioned GMRES(10) iterations to reduce the relative residual by a factor of 10^{-6}.

n	m	P_D	P_1	P_2	P_3	P_4
450	225	0.17	0.020	-	0.12	0.015
1922	961	1.19	0.083	-	1.58	0.27
7938	3969	20.3	0.37	-	33.4	8.70
32258	16129	214	1.83	-	622	324

Table 4: Time (in seconds) for preconditioned GMRES(10) to reduce the relative residual by a factor of 10^{-6}.

5 Conclusions

We conclude by observing that our results demonstrate the effectiveness of using non-symmetric constraint preconditioners. By careful selection of the constraint preconditioner, we have shown that they can be applied in an operator-based manner either by using very simple choices of G or by using an implicit-factorization. The next step will be to incorporate these preconditioners into BOUT++ and Nektar++ [9] to see how they perform within a non-linear simulation.

References

[1] V. Alexandrov, A. Lebedev, E. Sahin, and S. Thorne. Linear systems of equations and preconditioners relating to the NEPTUNE Programme: a brief overview. Technical Report 2047353-TN-02, UKAEA, 2021.

[2] O. Axelsson and M. Neytcheva. Preconditioning methods for linear systems arising in constrained optimization problems. *Numerical linear algebra with applications*, 10(1-2):3–31, 2003.

[3] L. Bergamaschi, J. Gondzio, and G. Zilli. Preconditioning indefinite systems in interior point methods for optimization. *Computational Optimization and Applications*, 28(2):149–171, 2004.

[4] H. S. Dollar, N. I. Gould, W. H. Schilders, and A. J. Wathen. Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems. *SIAM Journal on Matrix Analysis and Applications*, 28(1):170–189, 2006.

[5] B. Dudson, P. Hill, and J. Parker. BOUT++. online repository, 2020. URL `http://boutproject.github.io`.

[6] C. Keller, N. Gould, and A. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM J. Matrix Anal. Appl.*, 21:1300–1317, 2000.

[7] T. Rees and A. J. Wathen. Optimal solvers for pde-constrained optimizationchebyshev semi-iteration in preconditioning for problems including the mass matrix. *Electronic Transactions on Numerical Analysis*, 34, 2008.

[8] T. Rees, H. S. Dollar, and A. J. Wathen. Optimal solvers for pde-constrained optimization. *SIAM Journal on Scientific Computing*, 32(1):271–298, 2010. doi: 10.1137/080727154. URL `https://doi.org/10.1137/080727154`.

[9] S. Sherwin, M. Kirby, C. Cantwell, and D. Moxey. Nektar++. online, 2021. URL `https://www.nektar.info`.

Appendix A: Matlab Script

```
% Generates non-symmetric implicit-factorization constraint preconditioner
% families
format compact
Ll = [sym('l11'),sym('l12'), sym('l13'),sym('l21'),sym('l22'),...
    sym('l23'),sym('l31'),sym('l32'),sym('l33') ];
Rr = [sym('r11'),sym('r12'), sym('r13'),sym('r21'),sym('r22'),...
    sym('r23'),sym('r31'),sym('r32'),sym('r33') ]
Mm = [sym('m11'),sym('m12'), sym('m13'),sym('m21'),sym('m22'),...
    sym('m23'),sym('m31'),sym('m32'),sym('m33') ]
total = 0;

for i=1:5
    for j=1:3
        for k=1:5
            L = [Ll(1:3);Ll(4:6);Ll(7:9)];
            R = [Rr(1:3);Rr(4:6);Rr(7:9)];
            M = [Mm(1:3);Mm(4:6);Mm(7:9)];
            switch i
                case 1
                    L(1,1:2)=0; L(2,1)=0;
                case 2
                    L(1,1:2)=0; L(3,2)=0;
                case 3
                    L(1,2)=0; L(3,1:2)=0;
                case 4
                    L(2,1)=0; L(3,1:2)=0;
                case 5
                    L(1,2)=0; L(3,2:3)=0;
            end

            switch j
                case 1
                    M(3,2:3)=0; M(2,3)=0;
                case 2
                    M(1,1:2)=0; M(2,1)=0;
                case 3
                    M(1,2)=0; M(2,1)=0; M(2,3)=0; M(3,2)=0;
            end

            switch k
                case 1
                    R(1:2,1)=0; R(1,2)=0;
                case 2
                    R(1:2,1)=0; R(2,3)=0;
                case 3
                    R(2,1)=0; R(2:3,3)=0;
                case 4
                    R(2,1)=0; R(1:2,3)=0;
                case 5
                    R(1,2)=0; R(1:2,3)=0;
            end
            p = 1;
            F = L*M*R;
            if ((F(1,3)==0) | (F(2,3)==0) | (F(3,1)==0) | (F(3,2)==0))
```

```
                p=0;
            end

            if (p==1)
                total = total+1;
               % [i,j,k]
               factor = total
                struct=[L,M,R]
                F
            end
        end
    end
end
total
```