libcKrylov: A light weight linear solver library in C

Pengliang Yang
School of Mathematics, Harbin Institute of Technology, 150001, Harbin
E-mail: ypl.2100@gmail.com

July 30, 2023

Abstract

Solution of the linear system is of fundamental importance in science and engineering applications, which can be done by direct Gaussian elimination or iterative methods in Krylov space. We deliver a light weight library `libcKrylov` for iterative solution of linear system using C programming. The solvers include conjugate gradient (CG) method, BiCGStab and GMRES and its preconditioned variants. The construction of `libcKrylov` follows the reverse communication style, allowing the system specified by a sparse banded matrix or a matrix-free operator. The functionalities of the solvers are illustrated by the unsymmetric matrix inversion, and solving the large, sparse linear system after discretization of partial differential equations (the frequency domain wave equation and the Poisson equation).

1 Motivation and significance

Solution of a linear system is a ubiquitous problem, thus of fundamental importance in science and engineering applications. Mathematically, it can simply be formulated as

$$Ax = b, \tag{1}$$

where $A \in \mathcal{C}^{m\times n}$ is a matrix of m rows and n columns; the vector $x \in \mathcal{C}^n$ collects all the unknowns, sometimes referred to as model parameters; $b \in \mathcal{C}^m$ is the observation vector. For a well-posed problem with $m = n$ and an invertible A, an intuitive and brute-force approach is direct Gauss-elimination, via lower-upper (LU) triangular matrix decomposition, whose computational complexity is of the order $O(n^3)$. For small and medium size problem, such approaches are accurate and efficient. However, the computation and memory consumption of LU factorization easily reaches its bottleneck in 3D large-scale problems, making it infeasible to deal with practical issues. The solution of partial differential equations (Maxwell equation, wave equations, Navier-Stokes equations etc) represent a large class of such examples, yielding sparse and large linear systems to be inverted (**?**).

Consequently, the iterative methods become nice and handy alternatives to address such difficulties. Besides the relaxation methods such as Jacobi and Gauss-Seidel iterations and its accelerated variants, most of the effective iterative methods are working in Krylov space. This motivates the development of `libcKrylov` toolbox. In connection to the functionalities of `libcKrylov`, a number of well known computing packages such as PETSc (**?**) and Trilinos (**?**) have been developed which includes all kinds of linear solvers to deal with all kinds of computational challenges in numerical analysis. Let us point out that `libcKrylov` is never to be new to fulfill the same goal. When developing codes for a number of interesting geophysical applications such as seismic and electromagnetic modelling and inversion, the numerical modelling are often task independent so frequent data communications and sophisticated parallelism are seldomly used. `libcKrylov` finds a niche to avoid heavy dependency of our code on a number of other irrelevant software packages that PETSc must install. The rigid syntax requirement used by Krylov solver such as PETSc can be avoided in our own applications with `libcKrylov`.

2 Software description

2.1 Methods

Let us briefly review the numerical methods implemented by `libcKrylov` following **?**. The key to these numerical methods in Krylov space relies on the fact that finding the n unknowns from m data points in (**??**) is equivalent to solving a linear inverse problem.

When the matrix A is symmetric positive definite (SPD), the most favorable iterative technique for solving equation (**??**) is the conjugate gradient (CG) method. It is easy to shown that (**??**) is normal equation of the minimization objective functional

$$J(x) = \frac{1}{2} x^H A x - x^H b, \tag{2}$$

where ^{H} denotes complex conjugate transpose.

There are a number of other methods when the matrix A is not SPD. A simple idea is to minimize the residual of the equation in least-squares sense

$$J(x) = \frac{1}{2} \|Ax - b\|^2. \tag{3}$$

The normal equation of the above error functional, specified by first order optimality condition $\partial J / \partial x = 0$, gives

$$A^H A x = A^H b. \tag{4}$$

This system has a SPD coefficient matrix $A^H A$ so that it can be solved again using CG iterations. Such an approach is referred to as CGNR method. A slight different view point leads to a different method coined CGNE, by replacing x with $x := A^H y$ in equation (**??**) and solving for y

$$AA^H y = b, \tag{5}$$

in which AA^H is again an SPD matrix. Note that CGNR achieves minimal residual while CGNE achieves minimal error (defined as the error between numerical solution x^k and the true solution x^*, which is often unknown). Both CGNR and CGNE methods squares the condition number of the coefficient matrix, thus, these methods work well when the condition number of the matrix, $cond(A)$, is not very large.

When the condition number of the linear system in (**??**) is very large, the other two transpose free methods, biconjugate gradient stabilized (BiCGStab) version (**??**) and generalized minimal residual (GMRES) (**?**) methods, are often the method of choices. GMRES ensures a monotonic decreasing trend of the equation residual, as long as sufficient computater memory is available to generate new vectors which is orthogonal to all previous projection vectors. As memory is always an important concern in solving large scale problems, a restart strategy has been adopted, which may lead to the stagnation of the residual. BiCGStab does not guarantee the residual vector after each iterate has a decreasing norm, however, it avoids the memory overhead of GMRES method.

Given a highly ill-conditioned problem, all the above algorithms may fail to converge within reasonable number of iterations. In such cases, a preconditioner must be designed to reach convergence. Even if the system is not close to singular, a good choice of preconditioner often makes the algorithms converge faster.

2.2 Software structure

The software - `libcKrylov` - consists of the following components:

- `src`: This folder collects all detailed implementation of many algorithms in Krylov space. These are CG, CGNE, CGNR, BiCGStab, GMRES as well as their preconditioned version. The algorithm in complex arithmetics are prefixed with a letter 'c'. A number of other algorithms such as FOM, BCG, QMR, as well as preconditioned CGNR and CGNE emthods are not included. These are important intermediate variants to more advanced algorithms such as BiCGStab and GMRES. In addition, preconditioning for CGNE and CGNR requires approximating AA^H and $A^H A$, which is usually challenging and scarely used in practice.

- include: There are two header files solver.h and csolver.h, corresponding to real (double floating point) and complex arithmetics.
- lib: This is the directory that a static library file, after compliation with 'make', will be created here, i.e., libcKrylov.a.
- doc: This folder contains a detailed presentation of the pseudocode associated with the algorithms implemented in src, where a number of important references are listed.
- demo_poisson, demo_wave, and demo_toeplitz: These are the application examples presented in the next section of this article. They should be considered as template for the use of libcKrylov.
- README.md and LISENCE.txt.

Figure **??** provides a schematic diagram of libcKrylov in a tree structure.

The software must be compiled with C compiler, using Makefile. In order to use the library libcKrylov, one must first link libcKrylov.a when compling his/her own code by #include "solver.h" (or csolver.h for complex arithmetics). Typically, consider adding the following lines in his/her own Makefile:

```
libcKrylov_INC = ../include
libcKrylov_LIB = ../lib

LIB = -L$(libcKrylov_LIB) -lcKrylov -lm
INC = -I$(libcKrylov_INC) -I.
```

A good demonstration is the 'Makefile' for the 'illustrative examples'.

2.3 Recommended coding paradigm for linear operator

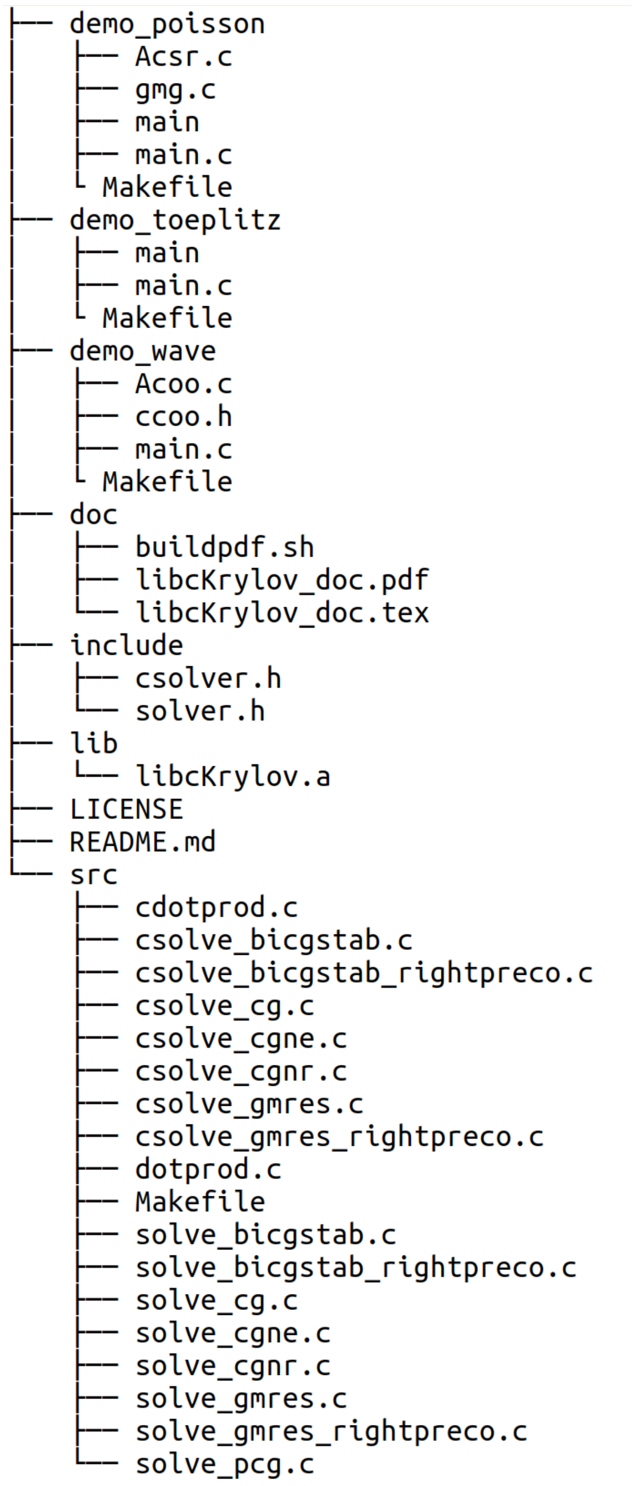
libcKrylov toolbox can be used consistently by adding three parts of the linear solver for given problem:

- Aop_apply(...) and/or possibly Atop_apply(...); We emphasis that to be consistent with all different linear solvers in libcKrylov, there are only 3 input parameters allowed in the linear operator, according the definition in solver.h:

  ```
  typedef void (*op_t)(int, double*, double*);
  ```

  where op_t is the placeholder for the name of the linear operator. This means an integer n indicating the length of the vector, the input vector x[] and the output vector y[] should be included in Aop_apply(n, x, y) and Atop_apply(n, x, y).
- Aop_init(...); It is clear that for most of the problems at hand, we would need a number of other important parameters. These parameters cannot be prescribed in the previous three in the linear operator. This can be done by an initialization step using Aop_init(...), prior to the use of Aop_apply(n, x, y) and Atop_apply(n, x, y). In this step, we should allocate memory for relevant arrays, which may be visible by Aop_apply(n, x, y) and Atop_apply(n, x, y) internally in terms of the scope of the variables. Therefore, this function is essentially a constructor.
- Aop_close(...); After all computations have been completed using iterative solvers, we use the routine Aop_close(...) to free all allocated memories. As a result, this function is essentially a destructor.

Note that an array allocated by Aop_init(...) deallocated by Aop_close(...) ensures the code is memory safe, if Aop_init and Aop_close(...) are called at the beginning and the end of the program. The above three key subroutines ensure that the functionality of the linear operator is a self-contained module. Since a preconditioner, in essence, is still a linear operator, the same convention should be adopted for coding a preconditioner. In fact, this design is no thing than the reverse communication (**?**) widely used by a number of Fortran codes (ARPACK for example). It is recommended to create a clean and efficient solver.



```
├── demo_poisson
│   ├── Acsr.c
│   ├── gmg.c
│   ├── main
│   ├── main.c
│   └ Makefile
├── demo_toeplitz
│   ├── main
│   ├── main.c
│   └ Makefile
├── demo_wave
│   ├── Acoo.c
│   ├── ccoo.h
│   ├── main.c
│   └ Makefile
├── doc
│   ├── buildpdf.sh
│   ├── libcKrylov_doc.pdf
│   └── libcKrylov_doc.tex
├── include
│   ├── csolver.h
│   └── solver.h
├── lib
│   └── libcKrylov.a
├── LICENSE
├── README.md
└── src
    ├── cdotprod.c
    ├── csolve_bicgstab.c
    ├── csolve_bicgstab_rightpreco.c
    ├── csolve_cg.c
    ├── csolve_cgne.c
    ├── csolve_cgnr.c
    ├── csolve_gmres.c
    ├── csolve_gmres_rightpreco.c
    ├── dotprod.c
    ├── Makefile
    ├── solve_bicgstab.c
    ├── solve_bicgstab_rightpreco.c
    ├── solve_cg.c
    ├── solve_cgne.c
    ├── solve_cgnr.c
    ├── solve_gmres.c
    ├── solve_gmres_rightpreco.c
    └── solve_pcg.c
```

Figure 1: The tree structure of libcKrylov

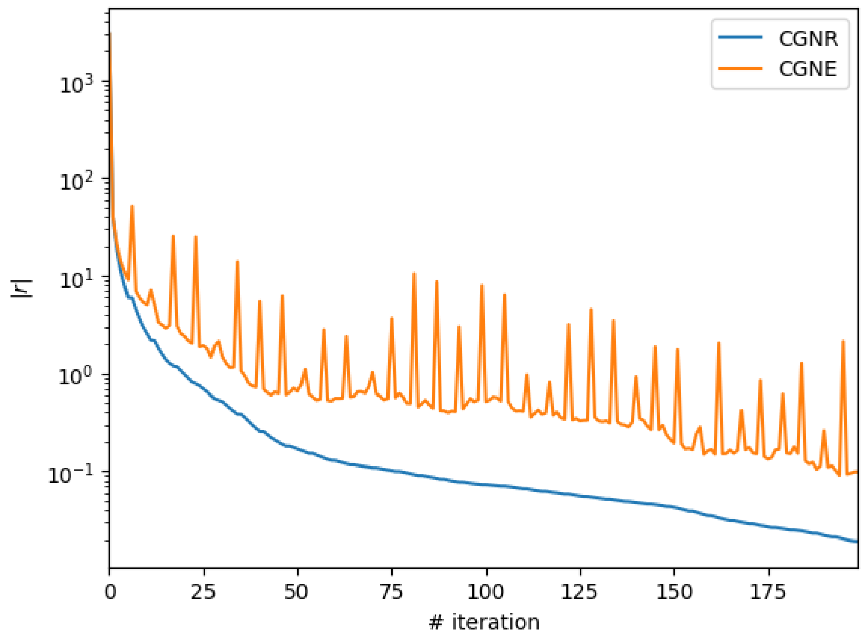


Figure 2: The convergence history of CGNR and CGNE within 200 iterations

3 Illustrative examples

3.1 Solution of a Toeplitz matrix system

Our first example is the inversion of an unsymmetric Toeplitz matrix A, in which $A_{ij} = r_{i-j}$ corresponding to $2n-1$ different scalars $r_{-n+1}, \cdots, r_0, \cdots, r_{n-1}$. Let us remark is that we do not need to store the whole Toeplitz matrix which is dense, but a vector r with the distance between row and column indices are sufficient to represent all elements of Toeplitz matrix. We choose $n = 500$ and generate $2n-1 = 999$ random numbers for $r_{-n+1}, \cdots r_{n-1}$. This creates an invertible matrix with relatively large condition number, allowing us to explore the performance of CGNR and CGNE methods within 200 iterations without reaching the tolerance of 10^{-6}. Note that both the matrix and its transpose applied to a vector are required in CGNR and CGNE methods. As shown in Figure **??**, both CGNR and CGNE have a good convergence trend. CGNR is decreasing the norm of the residual monotonically, but CGNE is not. This is consistent with the theory, as CGNE is minimizing the error between the numeric solution and the true solution, instead of minimizing the residual of the equation.

3.2 Frequency-domain wave equation

The second example is to solve a frequency domain wave equation (Helmholz equation) of the following form

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right) p(x,z,\omega) + \frac{\omega^2}{v^2(x,z)} p(x,z,\omega) = S(x,z,\omega) \tag{6}$$

where ω is the angular frequency and x, z are spatial coordinates, $v(x,z)$ is the wave speed, $p(x,z,\omega)$ is the pressure, $S(x,z,\omega)$ is the source term.

We discretized the wave equation using 2nd order finite difference method (hence a frequency-domain finite difference method, often coined FDFD) in space. To use truncated domain mimicking the wave propagating to infinitely far distance, we employ the perfectly matched layer (PML) technique in (**?**). This implies that the waves are gradually attenuated in the boundary layers. The adding of PML gives a complex-valued coefficient matrix which is no more symmetric. This matrix are stored in complex numbers in triplet coordinate (COO) format (**?**).

Since the matrix is not Hermitian symmetric, we use BiCGStab and GMRES algorithms in complex arithmetics to solve the resulting linear system. Figure **??** shows the iterative solution of BiCGStab and GMRES after 100 and 300 iterations. With the increase of the number of iterations, we see that the time harmonic wave solution is expanding spatially. However, one has to pay attention to the convergence of these two algorithms in resolving Helmholz equation. According to Figure **??**, the GMRES with restart=10 yields a monotonic decreasing residual while BiCGStab is not. Note that both methods converges very slowly, indicating an effective preconditioner is really needed.

Let us remark that solution of the wave equation is of fundamental importance in a number of geophysical applications, the problem has been continuously pursued and still remains an active area of research. Due to prohibitive large memory requirement of direct solvers, solution of 3D wave equation using iterative solver for complex inhomogeneous models is still an open question, despite a number of interesting preconditioners have been designed (**??**). However, designing a preconditioner dedicated to Helmholtz equation is out of the scope of this software paper.

3.3 Poisson's equation

The third example is the Poission's equation

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)u(x,y)=f(x,y), \tag{7}$$

where we use $f(x,y)=2(1-6x^2)y^2(1-y^2)+2(1-6y^2)x^2(1-x^2)$. Under the homogeneous boundary condition, the analytic solution of this equation can be explicitly found: $u(x,y)=(x^2-x^4)(y^2-y^4)$, as shown in Figure **??**a. Since the Laplacian operator is self-adjoint, it leads to a real-valued, SPD matrix system to be inverted. We store the Poisson matrix in compressed sparse row (CSR) format. We use 200 iterations to solve the Poisson equation with a tolerence of the relative error 10^{-6} to test the performance of CG method and its preconditioned version using multigrid V-cycle, following (**?**).

Figure **??**b and **??**c display the error between the true solution and the numeric solution found by CG and PCG. As can be seen from Figure **??**, PCG converges much faster than CG: it reaches the tolerence within 58 iterations, while CG after 200 iterations still has larger errors than PCG solution. This can be obviously seen by checking the magnitude of the error panels in Figure **??**b and **??**c.

One thing to remark is that `libcKrylov` allows a linear operator coded using compressed sparse matrix format, but also in a matrix free fasion. In this example, the geometrical multigrid preconditioner is in fact programmed in a matrix free manner, which saves the computer memory while remains efficient in implementation. On the other side, the example highlights the importance of an effective preconditioner used in iterative method to resolve the computational challenges.

4 Impact and future extensions

To date, the numerical methods to solve a linear system have been well established. `libcKrylov` serves as a slim and flexible code without the overhead of a number of irrelevant software packages and sophiscated data parallelism, solving the linear system using various number of Krylov space iterative methods. Equipped with a C compiler, one can benefit from a number of mainstream linear solver, possibly combined with efficient preconditioners to address their own problem. This can be done by either linking the compiled library or directly copy relevant solver into their own project. Using `libcKrylov` toolbox, one can simply focus on developing the linear operator without the need to recreate these mature solvers.

The last two illustrative examples come from direct discretization of PDE using finite difference method, possessing a sparse banded coefficient matrix. In principle, most of the numerical methods, if not all, including finite volume method and finite element methods, lead to a large linear system, which may not necessarily be sparse. In this case, they can still be solved with `libcKrylov`: the matrix-free implementation is preferred, so that the action of the matrix applied to an input vector can be computed precisely. Based on the framework of `libcKrylov` toolbox, we are currently developing a 3D least-square reverse time migration code for seismic imaging, as well as a 3D controlled-source electromagnetic modeler using preconditioned iterative approaches.

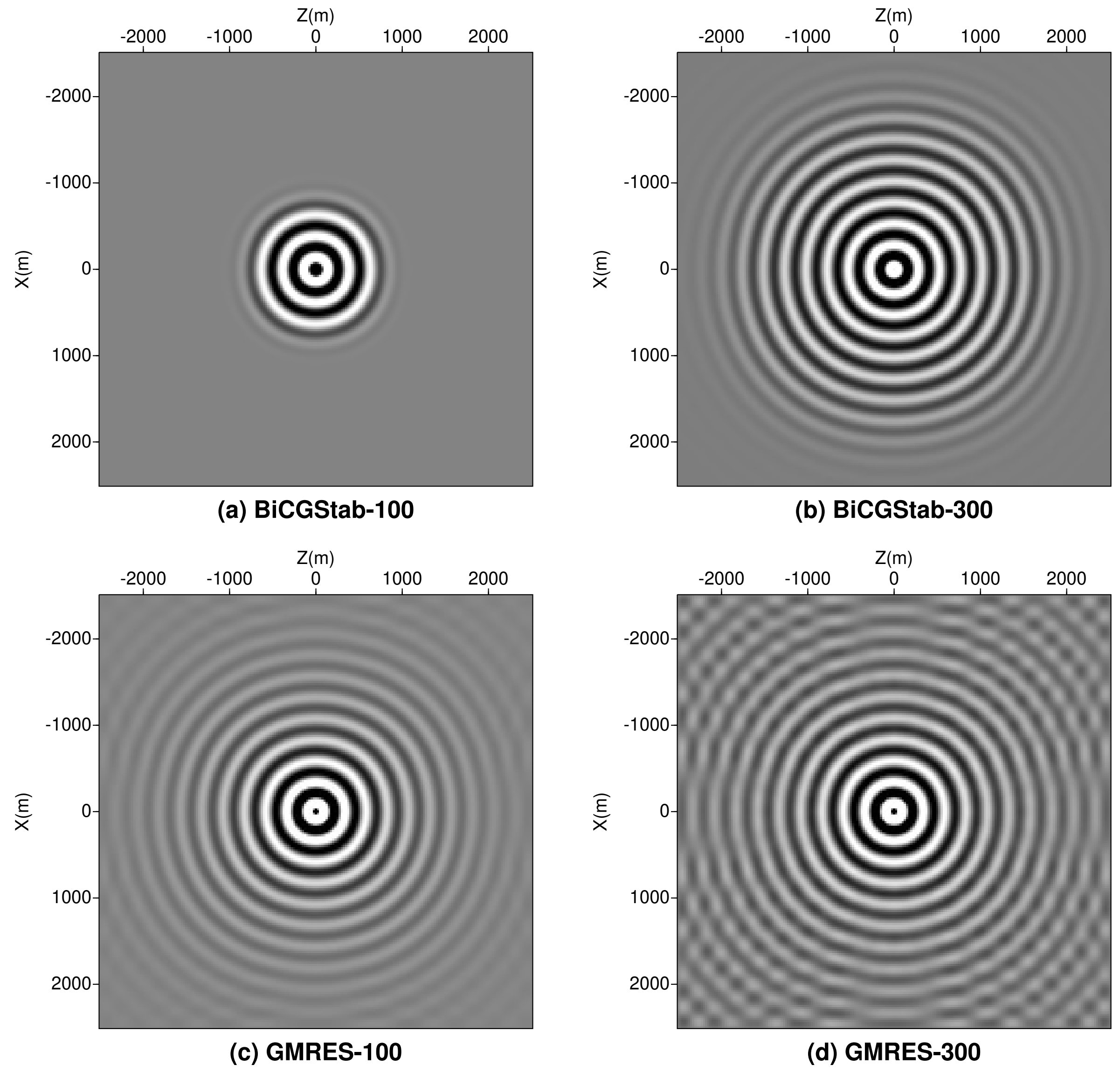


Figure 3: Frequency-domain wavefield computed by BiCGStab and GMRES (restart=10): (a) BiCGStab with 100 iterations; (b) BiCGStab with 300 iterations; (c) GMRES with 100 iterations; (d) GMRES with 300 iterations

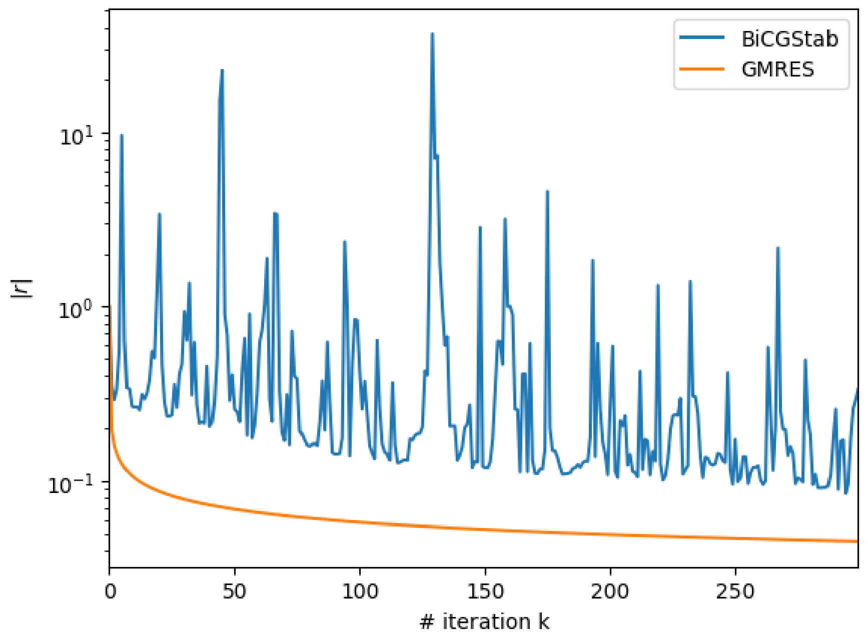


Figure 4: The convergence history of BiCGStab and GMRES within 300 iterations

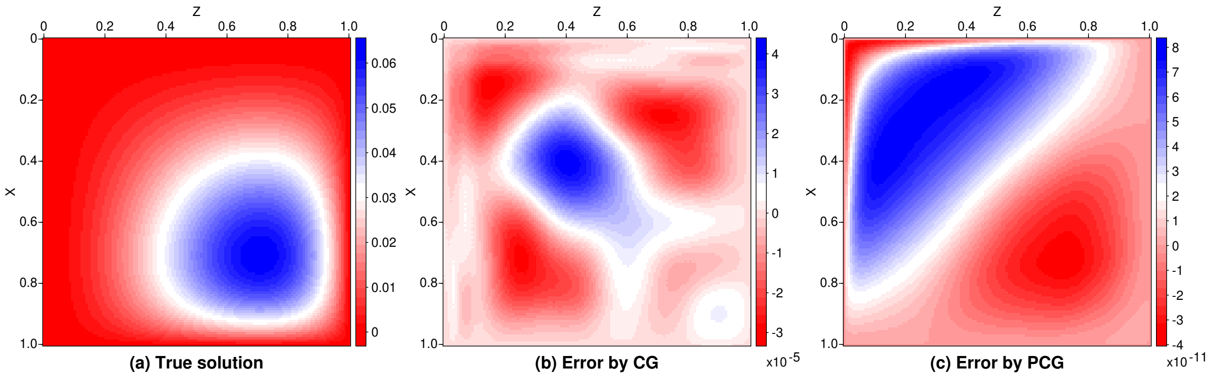


Figure 5: (a) The true solution u^*; (b) The error $|u - u^*|$ by CG; (c) The error $|u - u^*|$ by PCG.

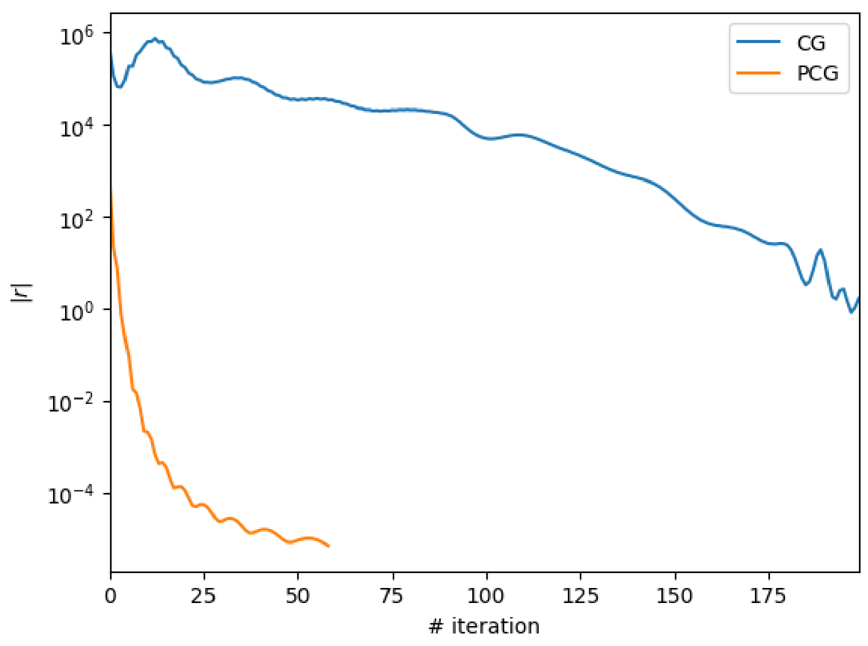


Figure 6: The convergence history of CG and PCG within 200 iterations

Current code version

C1	Current code version	v1.0
C2	Permanent link to code used for this code version	`https://github.com/yangpl/libcKrylov`
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	GPL v3.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	C
C7	Compilation requirements, operating environments & dependencies	Linux OS
C8	If available Link to developer documentation/manual	`https://github.com/yangpl/libcKrylov/doc/libcKrylov_doc.pdf`
C9	Support email for questions	

Table 1: Code metadata