Instruction for C++ package of visco-elastic multiparameter FWI in the frequency domain

Jinji Li, Scott Keating, Daniel Trad, and Kris Innanen

ABSTRACT

The frequency domain full-waveform inversion (FWI) is a nonlinear optimization problem where large matrices such as the Helmholtz matrix and many derivatives are saved and utilized. In many inversion tests involving multiple variables and larger-scale models, the computational cost and consumption of computer resources should be considered. Matlab is presumably the best platform for solving large linear systems, manipulating matrices, and intuitively analyzing the intermediate results. However, its massive memory occupation and limited computational effectiveness can hamper going to 3-D problems or some larger 2-D FWI. Motivated by this, we developed a C++ version of the current 2-D frequency domain multi-parameter visco-elastic FWI, which can be carried out on relatively larger models. The reduced cost also allows us to go to the 3-dimensional inverse problems or the probabilistic approaches. Essential tools and concepts for working within the C++ ecosystem are covered in this instructive report, with examples of how to use this package.

INTRODUCTION

Full-waveform inversion (FWI) recovers the realistically heterogeneous models of the interior earth by minimizing the discrepancies between measured and synthetic data modeled by the numerical solution of the equations of motion (Tarantola, 1984; Virieux and Operto, 2009; Fichtner et al., 2009). The nonlinearity of the FWI can be mitigated via hierarchical multiscale inversion strategies, which first exploit longer, gradually shorter wavelength components of the models. Thus a macro model that provides the kinematic information can be acquired in the early stages of the inversion to improve the convergence. (Bunks et al., 1995; Brossier et al., 2009; Virieux and Operto, 2009; Keating and Innanen, 2020). The frequency domain FWI, which models the wavefields into several discrete frequencies, is more naturally fitting and favored due to its flexibility in specifying multiple ingredients in the data space and the ability to include frequency-sensitive parameters. The current Fourier domain modeling approach embeds directly solving linear systems with a prerequisite of building the large impedance matrix (Pratt, 1990; Brossier et al., 2010). Even though previous studies have exploited the computational resources discussing parallelism for better factorization (Brossier et al., 2009; Sourbier et al., 2009a,b), the memory-expensive nature of the frequency domain FWI is still conspicuous in larger problems since many matrices with considerable dimensionalities, such as the impedance and the Helmholtz matrices, should be saved throughout the process.

The current powerful, versatile, and intuitive CREWES frequency domain FWI MATLAB package (Keating and Innanen, 2022) is widely used by many researchers. This package, fitting perfectly with MATLAB's user-friendly ecosystems, has provided an ideal laboratory for prototyping, testing, and implementing the FWI algorithms. With symbolic computation quickly done, it performs extensive data analysis and visualization. The drawback of it comes along with the MATLAB nature. First, MATLAB is endogenously slower as an

interpreted language than lower-level ones such as Fortran, C, or C++. It generally takes more time to execute than other compiled languages. It is also hard to develop real-time applications with it in the industry. Additionally, fast computers with sufficient memory are always required while programming. This demand becomes more intensive when the frequency domain FWI is conducted on larger models because many immense dense and sparse matrices with double precision are kept. Indeed, users can identify memory requirements and apply techniques more efficiently (as optimized by Keating and Innanen (2022)) if memory usage is an issue. Still, it is treating the symptoms rather than the root cause. Another disadvantage, however, is usually ignored in academics, is the comparatively high financial cost of MATLAB.

C++ is known as a general-purpose programming language, which is widely used nowadays for competitive programming. It is one of the oldest and most effective languages and continues to dominate programming. The programs coded by this language can be compiled and run on many platforms, such as Linux, Windows, and Mactonish. As a lower-level language, it allows users to manipulate memory usage to enhance performance, which is imperative for coding up the frequency domain FWI more efficiently. Motivated by this, we have developed a C++ package of visco-elastic multiparameter FWI in the frequency domain. While being able to deal with larger problems where the matrices are heavier, the comparative run time is shortened.

This report introduces the fundamental ideas for this package, including the demand for larger models in the FWI, the primary programming structure, and the dependencies we used in building the matrix-based codes. We design the form of this report in an engineering and instruction-like style so that individual who uses this package can quickly get familiar with it and carry out inversion problems either on the local machine or the clusters. Further optimization and iterative update are still needed to make it more user-friendly and effective.

COMPUTATIONAL CONSIDERATIONS ON MATRIX VIEW

To form a major part of our motivation, we analyze the size and approximated memory consumption of the heaviest matrices throughout the modeling and inversion process. We start by introducing the notations for the forward modeling problem that discretizes the wavefield onto the designed grids, namely the 2-D frequency domain elastic wave equation:

$$\begin{cases} \omega^2\rho u_x + \frac{\partial}{\partial x}\left[\tilde{\lambda}\left(\frac{\partial u_x}{\partial x}+\frac{\partial u_z}{\partial z}\right)+2\tilde{\mu}\frac{\partial u_x}{\partial x}\right]+\frac{\partial}{\partial z}\tilde{\mu}\left(\frac{\partial u_z}{\partial x}+\frac{\partial u_x}{\partial z}\right)+f_x=0, \\ \omega^2\rho u_z + \frac{\partial}{\partial z}\left[\tilde{\lambda}\left(\frac{\partial u_x}{\partial x}+\frac{\partial u_z}{\partial z}\right)+2\tilde{\mu}\frac{\partial u_z}{\partial z}\right]+\frac{\partial}{\partial x}\tilde{\mu}\left(\frac{\partial u_z}{\partial x}+\frac{\partial u_x}{\partial z}\right)+f_z=0, \end{cases} \tag{1}$$

where ω is the angular frequency, ρ is the density, u_x and u_z are displacement components, and f_x and f_z are the source terms in the horizontal and vertical directions, respectively; $\tilde{\lambda}$ and $\tilde{\mu}$ are complex Lamé parameters, and are frequency-dependent and related to quality factors Q_p and Q_s. The matrix multiplications can represent this wave equation, thus making it an implicit finite-difference approach denoted by a system of linear equations shown in equation (2) (Marfurt, 1984; Hustedt et al., 2004; Operto et al., 2007).

$$\mathbf{S}\left(\omega\right)\mathbf{u}\left(\omega\right)=\mathbf{f}\left(\omega\right), \tag{2}$$

where $\mathbf{S}$ is the impedance matrix, $\mathbf{u}$ is the meshed displacement field, and the right-hand side denotes the source term.

The shape and volume of matrix $\mathbf{S}$ should be taken into consideration while evaluating the computational cost, as the forward-modeling-like actions are conducted many times in the frequency domain FWI. In the 2-D space, $\mathbf{S}$ consists of a block tridiagonal central band, and two block tridiagonal side bands; each element within one band is a 2 by 2 submatrix used to operate on the displacement vector $u_{i,j}$ (Pratt, 1990). Despite the boundary width, assuming the model has nx and nz nodal points in the horizontal and vertical directions, respectively, the dimensionality of $\mathbf{S}$ will be $2 \times nx \times nz$ by $2 \times nx \times nz$, and the nonzero values are $9 \times nx \times nz$.

The resolution of an extensive sparse system of linear equations, however, remains to be very costly for large multi-parameter problems (Sourbier et al., 2007; Operto et al., 2007). First, the impedance matrix is complex: attenuation introduces complex coefficients for the visco-elastic wave equation, and boundary conditions bring the imaginary part in the elastic cases. Being complex and large makes its storage expensive. Direct solvers like the LU decomposition are used to get the multi-source data efficiently, but the factorization of the matrix requires a considerable amount of computer memory. The default datatype for numbers in MATLAB is double, which corresponds exactly to IEEE 754 Double Precision. Those are 8-byte numbers; this size is doubled when complex numbers are stored. If individuals intend to perform complicated math such as linear algebra, they must use a floating-point class such as a double or single. The single class requires only 4 bytes for one number, but some operations limitations suggest a risk when using single precision while doing complicated math. For example, to speculate 1 discrete frequency of the data generated by 1 double-component source on a 2-D model with the size of 100×200, the computer will have to allocate approximately 0.23 Gigabytes to store the double and complex factors while operating on MATLAB. Second, although the multi-resolution provided by individual frequency components helps to mitigate the nonlinearity, the large bulk of $\mathbf{u}$ is always a problem. The dimensionality of the non-sparse $\mathbf{u}$ is $2 \times nx \times nz$ by ns by nf, where ns and nf are source and frequency numbers. Additionally, this FWI uses the adjoint state method that backpropagates the data residual to form an updating direction and the L-BFGS method to iteratively approximate the inverse of the Hessian matrix, meaning that the similar linear system in equation (2) will be solved many times (see Keating and Innanen (2020); Keating and Shor (2022) for details). Even though such operations are highly optimized in MATLAB, all of the abovementioned points echo the demand for a more judicious design of data types and a more inherently faster programming language.

C++ PACKAGE OF FREQUENCY DOMAIN VISCO-ELASTIC FWI

The leading information about our C++ package is contained in this section. We start by introducing the dependencies for matrix manipulation and multi-threading. The basic structure of our code is then presented. We took advantage of the object-oriented feature by using classes to improve the code reusability, scalability, and efficiency.

Prerequisites

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. It is a versatile tool that supports all matrix sizes and numeric types. Decompositions can also be achieved efficiently via Eigen so that complex linear systems can be resolved with reliable results. The Eigen API is clean and expressive. Thus, it is recommended if users are looking to work on C++ platforms with a Matlab style. It is also an option for programmers to create their own matrix object in C++ with operator overloading. However, this is far from optimal for writing individual matrix libraries in a production environment because it is time-wasting and awkward unless significant time is spent on optimizing it. The above considerations are the reasons for us to choose the Eigen library. It is actively developed and releases new versions frequently. More information and the latest release can be found on the official website: `https://eigen.tuxfamily.org/index.php?title=Main_Page`. The compiler supports up to version 3.4, standard C++03, and will be C++14 following this version.

Here are some links for a good starting example: `https://eigen.tuxfamily.org/dox/GettingStarted.html`. Eigen is easy to install as it does not need to be linked to any external library. Instead, the header files are included in the code for your program. With GCC, it is necessary to use the -I flag for the compiler to find the Eigen header files.

Another helpful link is the translation between Eigen and MATLAB `https://eigen.tuxfamily.org/dox/AsciiQuickReference.txt`, where MATLAB coders can find the reference with Eigen.

There are several data types we can use in this package: $Matrix$ for dynamic dense matrices; $RowVector$ for 1-dimensional vectors; $Array$ for 1-D arrays. Users can define them with integer, float, and double precisions. The $SparseMatrix$ class also provides many functions to facilitate the usage in memory-intensive applications. A specialized representation storing only the nonzero coefficients can reduce memory consumption with increased performance. Eigen also has feasible solvers that can be used in solving the linear equation system, for example, $ConjugateGradient$ for matrix-free context, and $SparseLU$ for direct resolution. Many third-party modules are also supported by Eigen, such as $UmfPack$, $SuperLU$, and $PardisoLU$. We would like to emphasize that Eigen is unlikely to beat MATLAB for directly solving systems of linear equations because the latter will directly call Intel's Math Kernel Library (MKL), which is heavily optimized and multi-threaded. Individuals can also configure Eigen to fall back to MKL for a similar performance. In our package, however, we applied the built-in $SparseLU$ for solving linear equations because we would like to reduce the redundancy of the dependencies.

We performed the factorization and most of the resolution phases in single precision. However, for some more extensive memory-intensive applications' usage, a specialized representation storing only the nonzero coefficients can reduce memory consumption vector product in the truncated Gauss-Newton method (also see (Keating and Innanen, 2020; Keating and Shor, 2022)) to guarantee the accuracy and then cast the matrix back to float to minimize the memory usage.

The parallel implementation of the frequency domain FWI is different from the time domain: frequency-wise for the former, while shot-wise for the latter. Parallelism should be based on a domain decomposition of the computational domain, meaning one thread should compute a subdomain of the wavefields for all sources. The ideal number of threads in the forward modeling should equal the number of frequency components. However, in the multiscale approach, updating the model within each frequency band depends on the result in the subsequent bands. Accordingly, we designate the inversion to be sequential throughout frequency bands while parallelized with subfrequencies within each band. We use OpenMP for parallelization because it is relatively easy to implement in C++. Detailed information can be found on the official website: `https://www.openmp.org`. Other dependencies include the Basic Linear Algebra Subprograms (BLAS) and the Linear Algebra package (LAPACK). The BLAS are routines that provide standard building blocks for performing basic vector and matrix operations. The LAPACK improves the linear algebra performance as much as possible by calling the BLAS. The following links are the user guide and latest releases: `https://netlib.org/blas/`, `https://netlib.org/lapack/`.

Basic structure

Following the classical workflow of the frequency domain FWI, we have divided the project into three major parts: set-up, modeling, and inversion. Accordingly, we designed three classes named $Setup\_par\_VE$, $Forward\_VE$, and $Inversion\_VE$. The schematic structure of our package is shown in Figure (1). $General/$ contains several declarations of external dependencies, mainly the *std* functions, the Eigen libraries, the OpenMP headers, and other general functions used multiple times for slicing and indexing the matrices.

Additionally, $Main/$ is where the main function $test\_Inv.cpp$ is located. There are just a few lines in the main function because most computations are done within the three classes, but the choreography of these three classes is clear: reading parameters, modeling, and inversion.

The first class reads and sets all the parameters, including the geometries, the frequency information, and the models. All the settable parameters are stored in $parameter.txt$ in the directory $data/$; users can modify this file with specific preferences. The full list of in $parameter.txt$ is as below:

While creating the objects of the $Setup\_par\_VE$ class, several internal instances are automatically called to settle the parameters and read the models. The true and initial models should also be saved in $data/$ with the binary format. The models should be organized in one-vector form and ordered by $\rho - V_P - 1/Q_P - V_S - 1/Q_S$. For example, if the geometry is 50×50, the input model vector should be $(50 \times 50 \times 5) \times 1$. Every time the models are changed, users should specify the new names in $parameter.txt$, with changing the nx and nz accordingly. All the source codes for this class are saved in $src/Setup/$.

The second class, $Forward\_VE$, is designated for modeling the wavefield. Objects from the class $Setup\_par\_VE$ should be granted when initializing the modeling objects, such that it can use the parameters defined in the previous phase. Users can call the public instance $Get\_D()$ to compute the wave propagation in the synthetic model, and the result is

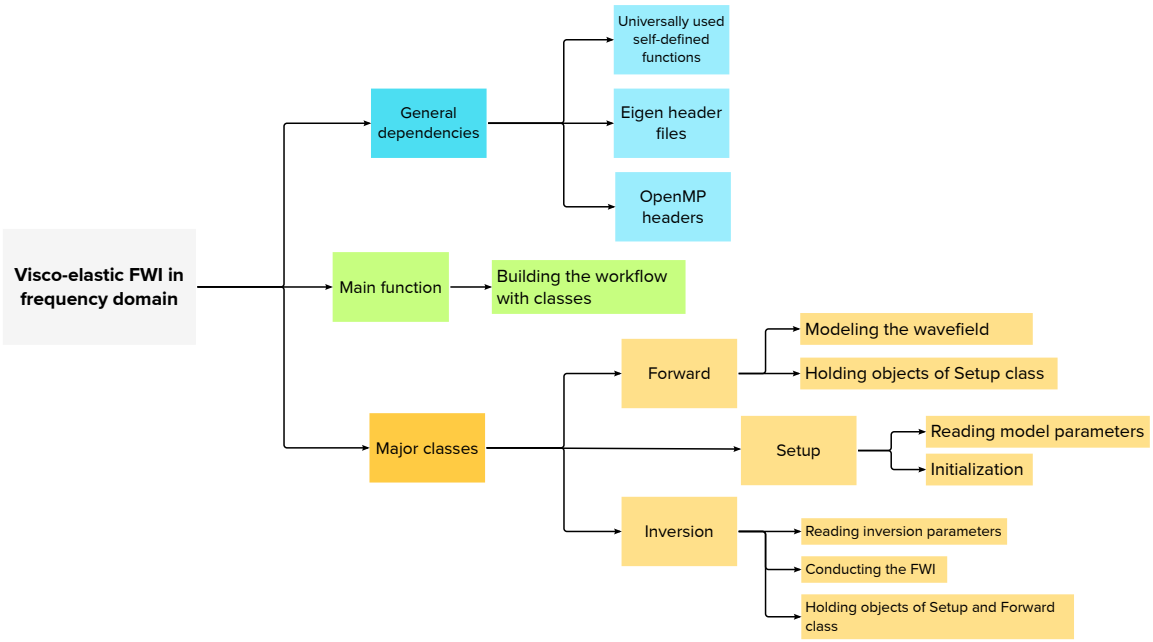

FIG. 1. Basic structure of the VEFWI package.

```
#include "../Setup/Setup_par_VE.h"
#include "../Forward/Forward_VE.h"
#include "../Inversion/Inversion_VE.h"

int main(){
    Setup_par_VE* this_par = new Setup_par_VE();
    Forward_VE* this_forward = new Forward_VE(this_par);
    this_forward->Get_D();
    Inversion_VE* this_inv = new Inversion_VE(this_par, this_forward
   );
    this_inv->Run_inversion();

    delete this_inv;
    delete this_forward;
    delete this_par;

    return 0;
    }
```

FIG. 2. Main function.

reachable as it is a public attribute called D. If users prefer to use real data, a new function has to be added to fulfill it. Figure (4) shows the pseudo-code describing the logic of the modeling phase. We currently use the ideal Dirac wavelet throughout the inversion process by defining a unique energy spectrum. Modifications on $fwave$ are necessary if a more realistic wavelet is required. All the source codes are saved in $/src/Forward/$.

The objects in the $Inversion\_VE$ class are initialized with objects in the previous two classes. It should hold all the model parameters and the synthetic data. However, some parameters, such as the iteration numbers, should be pre-assigned. The inversion

```
Line1: nx, nz - numbers of grid points in x and z directions
Line2: dx, dz - grid intervals in x and z directions
Line3: PML_thick - PML thickness
Line4: sAcq, rAcq - acquisition settings. Seperately for sources and
    receivers.
       coverage:
       1 - surface
       2 - left
       3 - right
       4 - bottom
       5 - top and left
       6 - top and right
       7 - top and bottom
       8 - right and left
       9 - all
Line5: soffset, roffset - offset of sources and receivers. This
   parameter sets where the array starts. The default values are
   zeros. Usually, it does not need to be changed.
Line6: P_grid, P_smooth - parameters for spacial Gaussian basis
   functions. The first value is P_grid, which is the spacing
   between basis functions in both x and z. The second one is
   P_smooth, and this is proportional to the size of the basis
   functions. Note: P_smooth does not reduce to the conventional
   expression when set to 0.
Line7: nP - to define the amount of inversion variables.
       means:
       1 - only density
       2 - density and VP
       3 - density, VP, and QP
       4 - density, VP, QP, and VS
       5 - density, VP, QP, VS, and QS
Line8: numbands, step - frequency parameters. The first value is the
    number of total bands, while the second one is how many
   subfrequencies withare in each band.
Line9: startband, endband1, endbandend - the first parameter is the
   lowest frequency of each band, the second parameter is the max
   frequency of the first band, and the last one is the max
   frequency of the last band.
Line10: Amp_scale - the amplitude scale.
Line11: f0 - the reference frequency.
Line12: file name of the true model. Note that the "./data/" should
   be kept.
Line13: file name of the initial model. Note that the "./data/"
   should be kept.
```

FIG. 3. List of parameters.

parameters in the file $inv\_par.txt$ within $data/$, as listed in Figure (5): This and the model parameter list are saved in the directory $data/$ with the suffix "info." As indicated above, this package implements the truncated Gauss-Newton method, and the inversion workflow is shown below: The inversion-related codes are saved in $src/Inversion/$.

The objectives (.o files) and dependence (.d files which Windows needs) are saved in the subdirectories under $obj/$ with the same name as source codes. An incremental compiling style is applied, which means only the modified and related dependencies will be re-compiled after the first time compilation.

```
#pragma omp for
for (int n = 0; n < freq.size(); n++){
    omega = 2 * pi * frequency(n);
    // compute current frequency.
    Make_Helm_Anelastic(model_parameters);
    // forming the Helmholtz-like matrix.
    Eigen::SparseLU<Eigen::SparseMatrix<std::complex<float>>> solver
    (A);
    // call and factorize the built-in sparseLU solver, which can be
    replaced by other reasonable solvers (UmpPackLU, PardisoLU, etc
    .).
    u = solver.solve((S * fwave(n)));
    // solve the linear system. fwave is the energy associated with
    the frequencies.
    U[n] = u;
    // save the wavefields into a vector.
    D[n] = R * U[n];
    // sample the wavefields to form D
}
```

FIG. 4. Pseudo-code of the modeling phase.

```
Line 1: optype - optimization type. Currently, we should only put 2
   because only the truncated Gauss-Newton approach is implemented.
Line 2: numits - number of outer iterations. Model in each band
   should be updated with this number.
Line 3: maxits - maximum iterations in approximating the inverse
   Hessian.
Line 4: tol - parameter that controls the optimization. This
   parameter does not to be changed in most scenarios.
Line 5: reg_fac - regularization factor. This parameter does not to
   be changed as well.
```

FIG. 5. List of inversion parameters.

```

for (int n = 0; n < frequency_band.size(); n++){
    freqs = frequency_band(n);
    // get the current frequency band.
    subD = D(freqs);
    // slice the dataset within the current band.
    for (int i = 0; i < outer_iter; i++){
        gradient = Gradient_VE(current model parameters, subD);
        // compute the gradient term. (OpemMP parallelized)
        descent_d = LBFGS_solve_linear(current model parameters,
    gradient);
        // compute the descent direction. Hessian-vector product is
    calculated many times (controlled by "maxits"). (OpemMP
    parallelized)
        alpha = Linesearch_Nocedal_Full(model parameters, descent_d)
    ;
        // get the updating step via line search.
        model = model + alpha * descent_d;
        // update the model in current band.
        }
}
```

FIG. 6. Pseudo-code of the inversion phase.

As mentioned in previous sections, the environment requirements of this FWI package are relatively simple. We use g++ to compile it on the Ubuntu platform. The compile options and commands are already integrated with the Makefile inside the $obj/$ folder, and users will need to run the "make" command inside this directory. After the compiling, a currently named executive file $test\_Inv.o$ will appear in the $obj/$ folder (the same location where Makefile is in), and users will run the inversion program with the command "$./test\_Inv.o$". The name of this objective file can be changed in the Makefile, but users must maintain the naming coincidence between the main function and this compiling target.

EXAMPLE

This part will elaborate on how this package is used in FWI experiments. We will go through the whole project, from compiling to running and show the critical codes with supplementary explanations in each step. As a quick guide for users, this part will mainly focus on the elaboration of how to successfully set the parameters and inputs such that this package can be directly applied without redundant edits.

Compilation

The package has been tested on the $Ubuntu$ Linux platform. For creating the executable, we use the utility $make$, which gets the instructions from a text file called, by default, $Makefile$. This utility invokes a compiler, a linker, and other dependencies, making an executable file. Re-compilation is needed every time for different computer architectures or source code changes. The main folder contains the $Makefile$ and the other subdirectories. The executables can be created by typing $make$ on the terminal or using an integrated development environment (IDE).

Before compiling, it is necessary to double-check if all the libraries are installed and linked. The $Makefile$ in this package is configurated for a standard Linux setup, but some changes may be needed. For example, the -I flag in line 19 should indicate the location of the $Eigen$ library. The $CCFLAGS$ and $LDFLAGS$ are compiling options. Generally, they don't need to be changed if the $OpenMP$, $LAPACK$, and $BLAS$ packages are installed correctly. Some compiling flags may vary in different systems. The first time $make$ is run, it takes the longest, about 2 minutes in our system, because all the functions need to be compiled and linked. This building step does not need to be rerun unless the source codes are modified. Below we see the information displayed during the building in our system: The name of the executable binary file is $test\_Inv$. Users can run the tests just by typing $./test\_Inv$.

Inversion test

Figure (8) illustrates the parameters used on the default inversion test. The list in Figure (3) can be referred to while changing this $parameter.txt$ file. Figure (9) shows the acquisition geometry on the synthetic model. The model size is 300 by 150 grid points in x and z directions, with a 20-meter interval. To simultaneously estimate the P-wave velocities and densities during the inversion, the parameter nP is set to 2 (see Figure (3), also Figure (8)). Figure (10) shows the true and initial models. The starting models result from smoothing

```
Building /home/jinji/VEFWI_class/Sur_FWI_CPP_new/obj/Main/test_Inv.o
Building /home/jinji/VEFWI_class/Sur_FWI_CPP_new/obj/General/
   intersect.o
Building /home/jinji/VEFWI_class/Sur_FWI_CPP_new/obj/Setup/
   Define_Acquisition_Explosive.o

                (skipping several similar lines)

Building /home/jinji/VEFWI_class/Sur_FWI_CPP_new/obj/Forward/
   Make_Helm_Anelastic.o

                (skipping several similar lines)

Building /home/jinji/VEFWI_class/Sur_FWI_CPP_new/obj/Inversion/
   FDFWI_VE.o
Linking test_Inv
```

FIG. 7. Information while compiling.

the true models by a Gaussian filter.

```
300 150
20 20
10
1 7
0 0
1 2
2
10 5
2 8 20
1
30
./data/model_true_Mar_100x50.dat
./data/model0_Mar_100x50.dat
```

FIG. 8. Parameter setting in the inversion test.

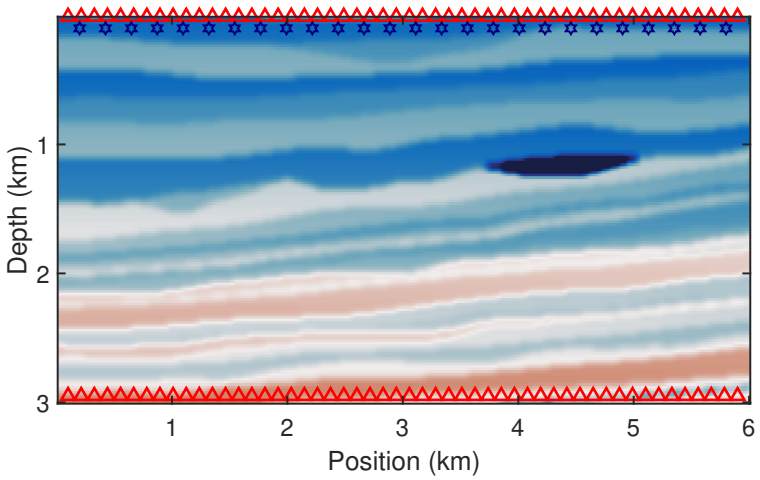

FIG. 9. Acquisition system in the inversion test. The dark-blue hexagrams are seismic sources, and the red triangles are receivers.

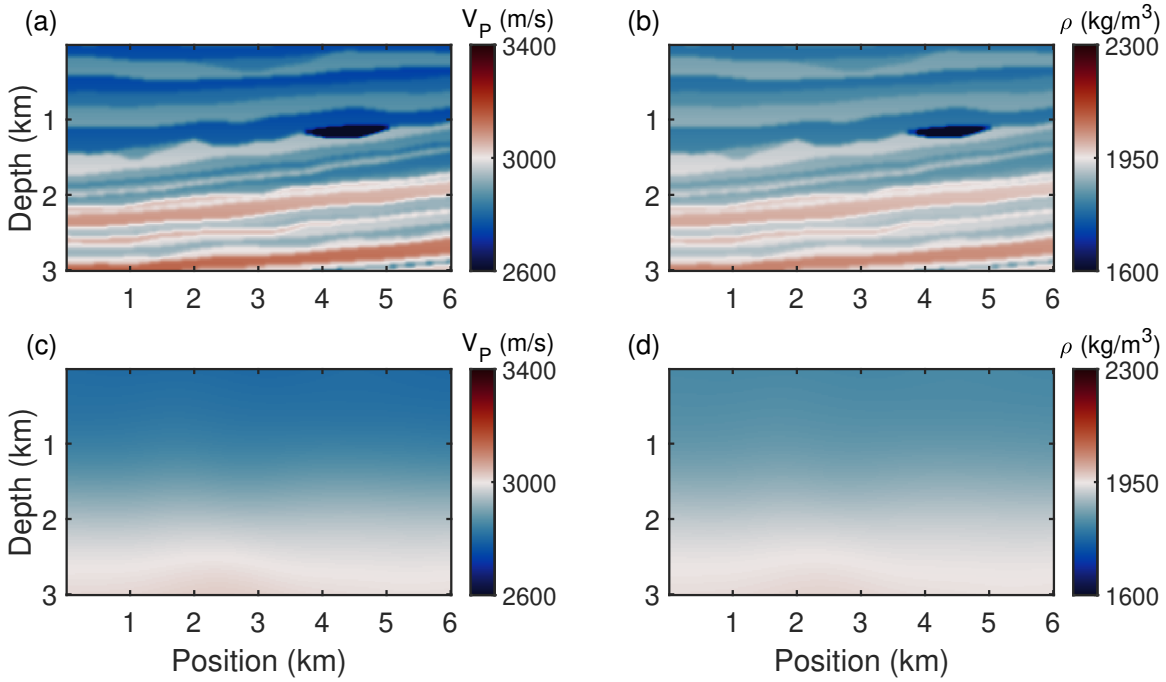

FIG. 10. True and initial models in the inversion test. (a), (c) true and initial P-wave velocity models. (b), (d) true and initial density models.

Usually, only line 2 and line 3 in $inv\_par.txt$ (Figure (5)) would be changed to set different inversion preferences. For example, in this test, we use 1 outer iteration and 20 inner iterations for the truncated Gauss-Newton method.

While running this code, the expected messages can be printed as Figure (11).

Once the FWI run completes, the output models are saved in files with the names $invmodel.dat$ in the $data/$ directory in $ascii - float$ format. They can be visualized using standard scientific plotting tools or seismic display tools, such as Seismic Unix or Madagascar. In this report, we use MATLAB to read and sketch the results. The saved vector's length will depend on the inversion variables specified by nP. For example, if nP is set to 2 as in the example, the output vector size will be $(nx \times nz \times 2) \times 1$, with the first $nx \times nz$ elements being ρ, and the rest $nx \times nz$ bulk being V_P values. For the example used in the validation, the models show a good definition of the subsurface structure, even though each band was updated only once.

```
Reading over.
*************************
nx =   300    nz =     150
dx =   20    dz =     20
PML_thick=       10
sAcq =   1    rAcq =     7
soffset =   0    roffset =     0
P_grid = 1
P_smooth = 2
numbands =   10    step =     5
startband =   2    endband1 =     8
endbandend =   20
Amp_scale =   1
Omega0 = 188.496
model_true = ./data/model_true_Mar_300x150.dat
model0 = ./data/model0_Mar_300x150.dat
*************************
Model done.
S num for one dimension: 149
R num for one dimension: 298
Acquisition defined.
P calculated.
Freq end.
Starting FD method.
U done.
The time for forward modeling is: 371.71s.
Ending FD method.
Reading inv par over.
Inversion starts.
Starting gradient.
Gradient done.
Starting optimization.
Relative residual: 0.21048
Relative residual: 0.13941

(skipping many similar lines)

Initial objective function is: 5.9607e-15
g: -3.5183e-16
g <= -c2 * g0.
Objective function after line search is: 1.6516e-15
Alpha: 1
=== Finished 1th outer iteration.

(many outer iterations)

Inversion ended.
Inv object destroyed.
Data object destroyed.
Parameter object destroyed.
```

FIG. 11. Information while running.

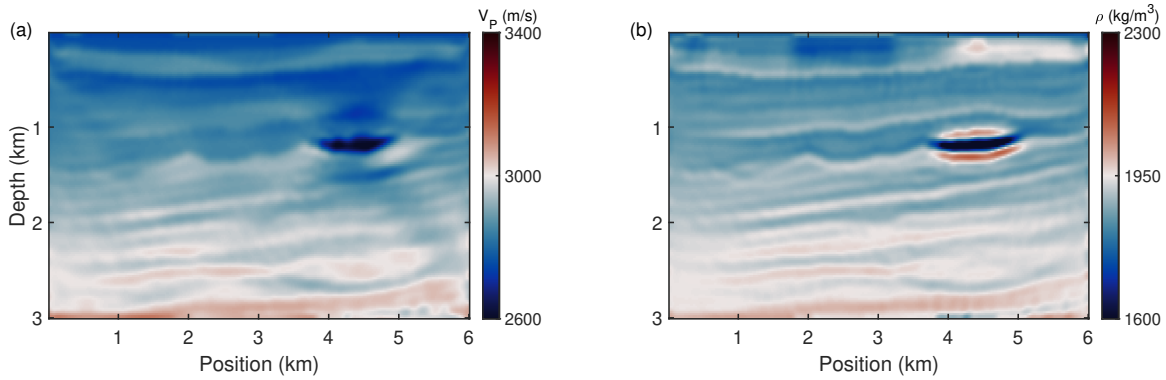

FIG. 12. Estimated models. (a) P-wave velocity. (b) Density.

COMPUTATIONAL EFFORT ANALYSIS

In this section, we examine the computational effort of the C++ visco-elastic FWI. We run multiple tests with different grid dimensions but the same physical properties, using an Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz with 12 threads. The runtime and memory consumption are shown in the two tables below.

Table 1: runtime measurement

Model size	Synthetic modeling	1 band inversion	Total runtime
50 × 50	6.23 s	58.1 s	581.3 s
100 × 50	15.31 s	162.93 s	1589.14 s
100 × 100	28.5 s	308.97 s	2097.9 s
150 × 100	57.59 s	655.6 s	6322.1 s
200 × 100	98.23 s	979.6 s	9701.43 s
250 × 100	143.76 s	1649 s	16351.6 s
300 × 100	215.45 s	1649 s	16351.6 s
300 × 150	371.71 s	3813.3 s	37017.1 s

Table 2: memory consumption

Model size	Synthetic modeling	Inversion
50 × 50	0.59 GB	1.47 GB
100 × 50	1.36 GB	2.52 GB
100 × 100	2.35 GB	4.51 GB
150 × 100	3.81 GB	8.25 GB
200 × 100	6.23 GB	11.9 GB
250 × 100	8.85 GB	15.1 GB
300 × 100	10.2 GB	20.8 GB
300 × 150	15.4 GB	26.2 GB

Our primary concern at this stage is the RAM usage and the speed, so we do not compare the numerical effect caused by different grid sizes. The total time in Table 1 is approximately the runtime for one band inversion multiplied by the overall frequency bands. The seconds for calculation are roughly doubled as the grid size enlarges by 50 in one dimension. As the grid dimension is gradually enlarged, the running time and memory consumption increase accordingly. Extending the grid horizontally not only magnifies the size of the impedance matrix but also extends the number of sources, both factors having a significant influence on the computational cost. Larger impedance matrices also demand larger RAM because more LU factors need to be stored while resolving equations. As previously mentioned, the inversion is carried out sequentially through frequency subsets while concurrently on individual components. For the parallelization, each thread holds a portion of the system of equations, including the blocky impedance matrix, to eliminate race conditions among tasks. However, there is a considerable overhead because of duplicated information, increasing the requirements for computer resources.

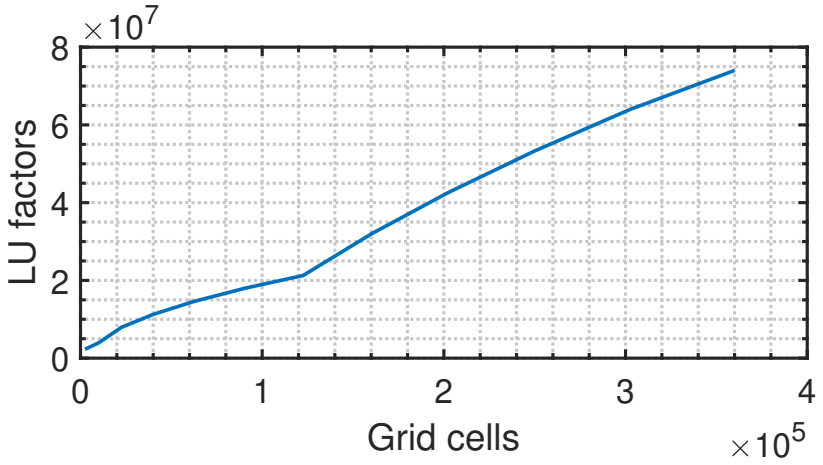

FIG. 13. Amount of LU factors as a function of the grid cell number.

CONCLUSIONS AND PERSPECTIVES

In this report, we presented a new implementation in C++ of a frequency domain viscoelastic FWI. The goal was to provide a memory-efficient and optimized performance that may be used for solving problems in large grids. When the model size increases, the high computational demand may prevent us from using the existing Matlab implementation. This report presents the first attempt to conduct efficient frequency domain FWI, which requires modeling a limited number of frequencies and hundreds to thousands of sources depending on the acquisition design. A simple text interface allows users to change parameters as needed. We provided an example to show how to operate the package and illustrate FWI results obtained with minimum configuration. More work is yet needed to make the program more productive and flexible.

In the present version, entire threads can only be exploited in the modeling phase because no communication is required across different frequencies. However, a more meticulous design is needed in the inversion process, as the subsequent results are always prerequisites at the beginning of each update. In such a situation, there are usually free workers

as long as the tasks within the current frequency band are fewer than the threads, indicating a possibility to optimize our program by creating inter-thread communications.

$SparseLU$, the built-in direct solver in the $Eigen$ library, works reasonably well throughout our prototyping and the numerical tests in this report. However, there is room for improvement by using more efficient solvers, such as MUltifrontal Massively Parallel sparse direct Solver (MUMPS) and the Intel MKL. An alternative for solving the forward problem is using hybrid direct/iterative methods relying on domain decomposition methods, although stability considerations require more research.

This C++ package was developed and tested on the VMware virtual system, which might affect performance since virtual function calls can slow down performance-critical applications. A virtual function call first needs to load the vtable pointer from the object and then load the function pointer from the vtable. These two steps can result in double data cache misses, dramatically reducing the performance in a tight loop.

As 3-D surveys become widely applied, corresponding 3-D techniques are required, which may be challenging because of their large amount of computation. The implementation in C++ we provided focuses on reducing the memory imprint, becoming a potential candidate for 3-D FWI problems.

A clumsy feature of this package is the lack of freedom for designing wavelet spectrum, and we plan to add such functions in further releases. The current version is available on GitHub (`https://github.com/JinjjLi/Multi-parameter_VEFWI_freq.git`), and the first author can be reached at `li.jinji@ucalgary.ca`. Any collaboration is more than welcome.

ACKNOWLEDGEMENTS

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 543578-19.

REFERENCES

Brossier, R., Etienne, V., Operto, S., and Virieux, J., 2010, Frequency-Domain Numerical Modelling of Visco-Acoustic Waves Based on Finite-Difference and Finite-Element Discontinuous Galerkin Methods.

Brossier, R., Operto, S., and Virieux, J., 2009, Seismic imaging of complex structures by 2d elastic frequency-domain full-waveform inversion: Geophysics, **74**.

Bunks, C., Saleck, F. M., Zaleski, S., and Chavent, G., 1995, Multiscale seismic waveform inversion: GEOPHYSICS, **60**, No. 5, 1457–1473, https://doi.org/10.1190/1.1443880.
URL https://doi.org/10.1190/1.1443880

Fichtner, A., Kennett, B. L. N., Igel, H., and Bunge, H.-P., 2009, Full seismic waveform tomography for upper-mantle structure in the australasian region using adjoint methods: Geophysical Journal International, **179**, No. 3, 1703–1725.

Hustedt, B., Operto, S., and Virieux, J., 2004, Mixed-grid and staggered-grid finite-difference methods for frequency-domain acoustic wave modelling: Geophysical Journal International, **157**, No. 3, 1269–1296, https://academic.oup.com/gji/article-pdf/157/3/1269/6056859/157-3-1269.pdf.
URL https://doi.org/10.1111/j.1365-246X.2004.02289.x

Keating, I. K. A. H., S., and Shor, R., 2022, Simultaneous waveform inversion of SWD data for P-wave velocity, density, and source parameters, CREWES Research Report, 34, 36.

Keating, S., and Innanen, K. A. H., 2020, Simultaneous recovery of source locations, moment tensors and subsurface models in 2D FWI, CREWES Research Report, 32, 33, 14.

Keating, S., and Innanen, K. A. H., 2022, User guide for the CREWES frequency domain FWI codes, CREWES Research Report, 34, 33.

Marfurt, K. J., 1984, Accuracy of finite-difference and finite-element modeling of the scalar and elastic wave equations: GEOPHYSICS, **49**, No. 5, 533–549.

Operto, S., Virieux, J., Amestoy, P., L'Excellent, J.-Y., Giraud, L., and Ali, H. B. H., 2007, 3d finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study: GEOPHYSICS, **72**, No. 5, SM195–SM211, https://doi.org/10.1190/1.2759835.
URL https://doi.org/10.1190/1.2759835

Pratt, R. G., 1990, Frequency-domain elastic wave modeling by finite differences: A tool for crosshole seismic imaging: GEOPHYSICS, **55**, No. 5, 626–632.

Sourbier, F., Operto, S., Virieux, J., Amestoy, P., and L'Excellent, J., 2007, A massively parallel frequencyâ€domain fullâ€waveform inversion algorithm for imaging acoustic media: Application to a dense OBS data set, 1893–1897, https://library.seg.org/doi/pdf/10.1190/1.2792860.
URL https://library.seg.org/doi/abs/10.1190/1.2792860

Sourbier, F., Operto, S., Virieux, J., Amestoy, P., and L'Excellent, J.-Y., 2009a, Fwt2d: A massively parallel program for frequency-domain full-waveform tomography of wide-aperture seismic data - part 1: Algorithm: Computers Geosciences, **35**, No. 3, 487–495.
URL https://www.sciencedirect.com/science/article/pii/S0098300408002689

Sourbier, F., Operto, S., Virieux, J., Amestoy, P., and L'Excellent, J.-Y., 2009b, Fwt2d: A massively parallel program for frequency-domain full-waveform tomography of wide-aperture seismic data - part 2: Numerical examples and scalability analysis: Computers Geosciences, **35**, No. 3, 496–514.
URL https://www.sciencedirect.com/science/article/pii/S0098300408002677

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: GEOPHYSICS, **49**, No. 8, 1259–1266.

Virieux, J., and Operto, S., 2009, An overview of full-waveform inversion in exploration geophysics: GEOPHYSICS, **74**, No. 6, WCC1–WCC26.