Solution of time-domain problems using Convolution Quadrature methods and BEM++

T. Betcke* t.betcke@ucl.ac.uk
N. Salles* n.salles@ucl.ac.uk
W. Smigaj w.smigaj@ucl.ac.uk

* Department of Mathematics, University College London
25 Gordon Street, London, WC1H 0AY, United-Kingdom

Keywords: boundary element method, time-domain problems, convolution quadrature.

Abstract

Convolution Quadrature methods are efficient techniques for the solution of time-domain wave problems in unbounded domains via Boundary Element Methods. In this proceeding we use the Convolution Quadrature approach to decouple the time-domain problems into a series of independent frequency-domain problems that can be solved efficiently in parallel. In contrast to previous approaches we solve many more frequency-domain problems than there are time steps. We demonstrate numerically that this approach approximates the underlying time-stepping scheme with exponential accuracy as the number of frequency problems is increased. The implementation of the method is done using BEM++, a modern C++ based boundary element library with an easy to use Python interface.

1 Introduction

Convolution Quadrature (CQ) is an efficient technique for the solution of time-domain wave problems via Boundary Element Methods. A well known interesting point is that CQ methods can be formulated in such a way as to obtain a number of independent frequency-domain problems, which can be easily solved in parallel. Typically, the number of frequency problems is chosen to be the same as the number of time steps. However, in this proceeding we demonstrate that by increasing the number of time steps one can achieve exponential convergence to the underlying time-stepping scheme that forms the basis of the Convolution Quadrature formulation.

For simplicity, in this proceeding we focus on multistep schemes. But similar approaches are also possible for Runge-Kutta based Convolution Quadrature formulations. The proceeding is organised as follows. We first introduce parallel CQ schemes based on multistep methods and propose an approach in which the number of frequency domain solves is decoupled from the number of time steps. We then discuss the numerical implementation of the frequency problems with BEM++, an open source C++ based boundary element library. It offers the Galerkin discretisation of Laplace, Helmholtz and Maxwell kernels, and can be accessed via an easy to use Python interface. We conclude the proceeding with a numerical example that shows the exponential convergence of the proposed method.

Let $\Omega \subset \mathbb{R}^3$ be an obstacle and $\Gamma = \partial\Omega$ its boundary. We define the exterior domain where we solve the wave equation by $\Omega_e = \mathbb{R}^3 \backslash \bar{\Omega}$ (see Figure 1). In this proceeding we consider the following acoustic problem:

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2}(x,t) - c^2 \Delta_x u(x,t) = 0, \ x \in \Omega_e, \\ u(x,0) = \dfrac{\partial u}{\partial t}(x,0) = 0, \\ u(x,t) = g(x,t) \text{ for } x \in \Gamma. \end{cases} \tag{1}$$

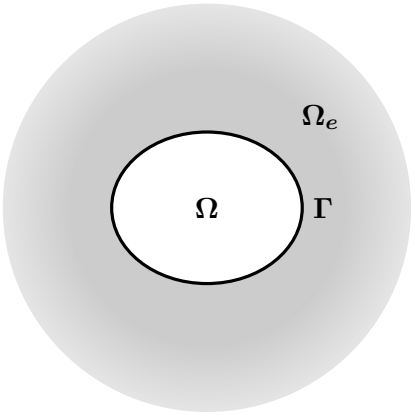

Figure 1: The exterior domain of interest Ω_e and the boundary of the scatterer.

2 Convolution Quadrature Methods

In this section we present a simple approach for the solution of (1) based on Convolution Quadrature. The idea is based on a Z-Transform of the time steps of an underlying time-stepping scheme. This leads to a range of modified Helmholtz problems in the frequency domain, which can be solved in parallel. The time domain solution is then synthesised by an inverse Z-transform. The proposed approach is similar to previous CQ methods (see e.g. [1, 2, 8]). However, in contrast to previous presentations we decouple the number of frequency solves from the number of time steps.

2.1 The Convolution Quadrature method

First, we transform the wave equation (1) into a first order system. To do this transformation, we introduce $v(x,t) = [u(x,t), \frac{1}{c}\frac{\partial u}{\partial t}(x,t)]^T$, $M = \begin{bmatrix} 0 & I \\ \Delta_x & 0 \end{bmatrix}$, $T = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$, and $B(x,t) = \begin{bmatrix} g(x,t), & 0 \end{bmatrix}^T$. We obtain

$$\begin{cases} \frac{1}{c}\frac{\partial v(x,t)}{\partial t} = Mv(x,t), \quad x \in \Omega_e, \\ v(x,0) = 0, \forall x \in \Omega_e, \\ Tv(x,t) = B(x,t), \quad x \in \Gamma. \end{cases} \tag{2}$$

Secondly, we apply a general multistep scheme of the form

$$\frac{1}{c\Delta t}\sum_{j=0}^{n} \gamma_{n-j} v_d(x,t_j) = Mv_d(x,t_n), \tag{3}$$

to obtain the discrete time step values $v_d(x,t_j)$ that approximate the exact values $v(x,t)$ at time steps $t_j, j = 0, \ldots$. We then apply the Z-transform:

$$\sum_{n=0}^{\infty}\left[\frac{1}{c\Delta t}\sum_{j=0}^{n}\gamma_{n-j}v_d(x,t_j)\right]z^n = \sum_{n=0}^{\infty} Mv_d(x,t_n)z^n. \tag{4}$$

The convolution becomes a product after the Z-Transform. We hence obtain

$$\frac{1}{c\Delta t}\gamma(z)V_d(x,z) = MV_d(x,z), \tag{5}$$

where $\gamma(z) = \sum_{n=0}^{\infty} \gamma_n z^n$ and $V_d(x,z) = \sum_{n=0}^{\infty} v_d(x,t_n) z^n$. For Backward Euler it holds that $\gamma(z) = 1 - z$. We now rewrite (5) as a modified Helmholtz equation in the frequency domain:

$$\begin{cases} \left(\dfrac{\gamma(z)}{c\Delta t}\right)^2 U_d(x,z) - \Delta U_d(x,z) = 0, \quad x \in \Omega_e, \\ U_d(x,z) = G(x,z), \quad x \in \Gamma, \end{cases} \tag{6}$$

where $U_d(x,z) = \sum_{n=0}^{\infty} u_d(x,t_n) z^n$ and $G(x,z) = \sum_{n=0}^{\infty} g(x,t_n) z^n$. Hence, for each given z we can evaluate $U_d(x,z)$ by solving the boundary value problem (6). The boundary conditions are obtained by a Z-transform of the time-domain boundary data. The time-domain solution is then obtained by the inverse Z-transform, which is given as a simple contour integral:

$$u_d(x,t_n) = \frac{1}{2\pi i} \int_{\mathcal{C}} \frac{U_d(x,z)}{z^{n+1}} dz, \tag{7}$$

with $\mathcal{C}$ a contour around the origin in the region of convergence of the Z-transform. Hence, we use $\mathcal{C} = \{z \in \mathbb{C} : |z| = \lambda\}$ for some appropriately chosen $\lambda > 0$. We approximate the integral using a trapezoidal rule with N_f frequencies at the points $z_k = e^{-2\pi i \frac{k}{N_f}}$ with $k = 1, \ldots, N_f$. This leads to

$$u_d(x,t_n) \approx u_d^{(N_f)}(x,t_n) := \frac{\lambda^{-n}}{N_f} \sum_{k=1}^{N_f} \frac{U_d(x, \lambda z_k)}{z_k^n}. \tag{8}$$

Note that it is possible to reduce the number of frequency problems to solve by approximatively 2 since

$$U(x, \overline{z}) = \overline{U(x,z)},$$

as noticed in [5, Subsection 4.1].

The accuracy of computing $u_d^{(N_f)}$ via (8) depends on the number N_f of frequency problems solved. In Section 4 we demonstrate that the rate of convergence is exponential.

3 Solving the frequency problems with BEM++

In this section we demonstrate how to implement the frequency-domain modified Helmholtz problems, using BEM++ [11]. BEM++ is an open-source boundary element library developed in C++ but with an extensive Python interface. It is fully object-oriented and can be easily extended. It makes heavy use of external projects, such as Trilinos [3] for the solution of linear systems, Dune [6, 7] for the grid management and implementation of basis functions on canonical elements, and TBB [4] for shared-memory parallelisation. Moreover, the library can make efficient use of AHMED 1.0 [10] for adaptive Cross Approximation (ACA) and Hierarchical Matrices ($\mathcal{H}$-matrices) algebra. Interfaces to external FMM libraries are in development.

The library currently supports Laplace, Helmholtz and Maxwell equations. Discretisation of kernels is done using a Galerkin approach, where the singular integrals are approximated by fully numerical singularity adapted quadrature rules (see e.g. [9]).

3.1 The frequency problem

In order to solve the wave equation, we have to be able to solve the modified Helmholtz equation:

$$\begin{aligned} &\Delta u(x) - k^2 u(x) = 0, x \in \Omega_e, \\ &u(x) = g(x), x \in \Gamma \\ &+ \text{Radiation condition when } |x| \to \infty. \end{aligned} \tag{9}$$

Various methods are possible but since we solve in a unbounded domain, boundary element methods are well adapted. The kernel associated to this equation is

$$G_\omega(x,y) = \frac{e^{-\omega\|x-y\|}}{4\pi\|x-y\|}. \tag{10}$$

A possibility is to use the following indirect second kind formulation:

$$\begin{gathered}\text{Find } \phi \in L^2(\Gamma) \text{ such as:} \\ \left(\frac{I}{2} + \mathcal{K}_\omega\right)\phi(x) = g(x), x \in \Gamma.\end{gathered} \tag{11}$$

where I is the identity operator and K is the double-layer boundary operator:

$$K_\omega\phi(x) = \int_\Gamma \frac{\partial G_\omega(x,y)}{\partial n_y}\phi(y)ds_y. \tag{12}$$

3.2 Python implementation

For the implementation, we first load the BEM++ library in Python.

```
import numpy as np
import sys
sys.path.append("~/bempp/python/")
from bempp import lib as bempplib
```

We added in the path the directory containing the BEM++ python library. Second, we read the mesh and we define the numerical quadrature strategy.

```
grid = bempplib.createGridFactory().importGmshGrid("triangular", "./sphere-h-0.1.msh")
quadStrategy = bempplib.createNumericalQuadratureStrategy("float64", "complex128")
options = bempplib.createAssemblyOptions()
context = bempplib.createContext(quadStrategy , options)
```

There exist various options to modify the default accuracy of the numerical quadrature rule. Here, we will not go into detail of this but rather refer to [11].

Since we approximate the solution in $L^2(\Gamma)$, we define piecewise constant basis functions and then the corresponding boundary layer operators.

```
pconsts = bempplib.createPiecewiseConstantScalarSpace(context , grid)
mass_matrix = bempplib.createIdentityOperator(context,pconsts,pconsts,pconsts)
dlOp = bempplib.createModifiedHelmholtz3dDoubleLayerBoundaryOperator(context,pconsts,
    pconsts,pconsts,wavenumber)
lhsOp = .5*mass_matrix + dlOp
```

Operators created in BEM++ take three spaces as arguments, the domain space, the range space, and the space dual to the range space. The range space is not strictly necessary for Galerkin discretisations, but allows BEM++ to automatically implement routines for an operator algebra, including the product of boundary integral operators.

At this level no matrix have been created. The following code creates a right hand side via user defined function, denoted by `evalDirichletData`.

```
rhs = bempplib.createGridFunction(context, pconsts, pconsts, evalDirichletData).
    coefficients()
```

In what follows the iterative solver is setup, and the problem solved. The `RealOperator` class from BEM++ turns a complex operator into an equivalent real operator. This is to circumvent a bug in the handling of complex matrices via Gmres in some commercial Python distributions, and may not be necessary depending on how Scipy was compiled. The actual discretisation of the operator takes place via the `weakForm` method, which returns a discretised operator that is compatible to the Scipy Operator interface.

```
from scipy.sparse.linalg import gmres
n=len(rhs)
A = RealOperator((.5*mass_matrix + dlOp).weakForm())
b = np.hstack([np.real(rhs), np.imag(rhs)])
sol_real,info=gmres(A,b,tol=1e-15,maxiter=1500)
sol=(sol_real[0:n]+1j*sol_real[n:]).reshape(n,1)
np.save('solution',sol)
```

To evaluate the solution in the domain, we first define a grid function by using the coefficients of `sol` in the space of piecewise constant basis functions. Then, a double layer potential is created to evaluate the solution at the `evaluation_points`:

```
evaluation_points=np.array([[0.,2.,0.],[0.,-2.,0.],[2.,0.,0.]])
gridFun=bempplib.createGridFunction(context,pconsts,coefficients=sol)
domain_solution=bempplib.createModifiedHelmholtz3dDoubleLayerPotentialOperator(context
    , wavenumber).evaluateAtPoints(gridFun,evaluation_points)
np.save('domain_solution',domain_solution)
```

Here, we evaluate the domain solution at only 3 points but it's possible to define a grid on a plan or in a volume to evaluate the domain solution.

4 Numerical result

In this section, we present a result for the acoustic scattering by a unit sphere, where the incoming wave is a plane wave localised in time by a Gaussian. The boundary condition in (1) writes as

$$g(x,t) = \cos\left(2\pi(t - \frac{k\cdot x}{c})f\right) e^{-\frac{(t-t_p-\frac{k\cdot x}{c})^2}{2\sigma^2}},$$

where f is the frequency, t_p the time-of-arrival, σ the variance, and k is the wave vector defining the direction of the incident wave. We use $f = 400, k = (1,0,0)^T, t_p = 0.01, c = 343$ and $\sigma = \frac{3}{1000\pi}$ for the computation.

Figure 2 presents the absolute difference $\left|u_d^{(N_f)} - u_d^{(\text{ref})}\right|$ according to N_f. The reference solution $u_d^{(\text{ref})}$ is obtained by choosing N_f very large. We can observe exponential convergence with a rate that depends on the parameter λ of the contour for the inverse Z-Transform. In [12] a full analysis of the asymptotic exponential rate of convergence for $N_f \to \infty$ is given. A particularly interesting point in Figure 2 is the point $N_f = N_t$, where the number of frequencies solved N_f is identical to the number of time steps N_t. This is the standard case in previous Convolution Quadrature approaches. We note that by increasing N_f a significantly better accuracy can be achieved.

Conclusion

We have presented an easy-to-use Convolution Quadrature method. By choosing the number of frequencies N_f larger than the number of time steps we can obtain a significantly improved accuracy. The solution of the frequency-domain problems with BEM++ has been presented. An open source time-domain toolbox for BEM++ is in development, which will allow the parallel solution of time-domain problems using various CQ approaches.

Acknowledgments. This work was supported by Engineering and Physical Sciences Research Council Grant EP/K03829X/1.

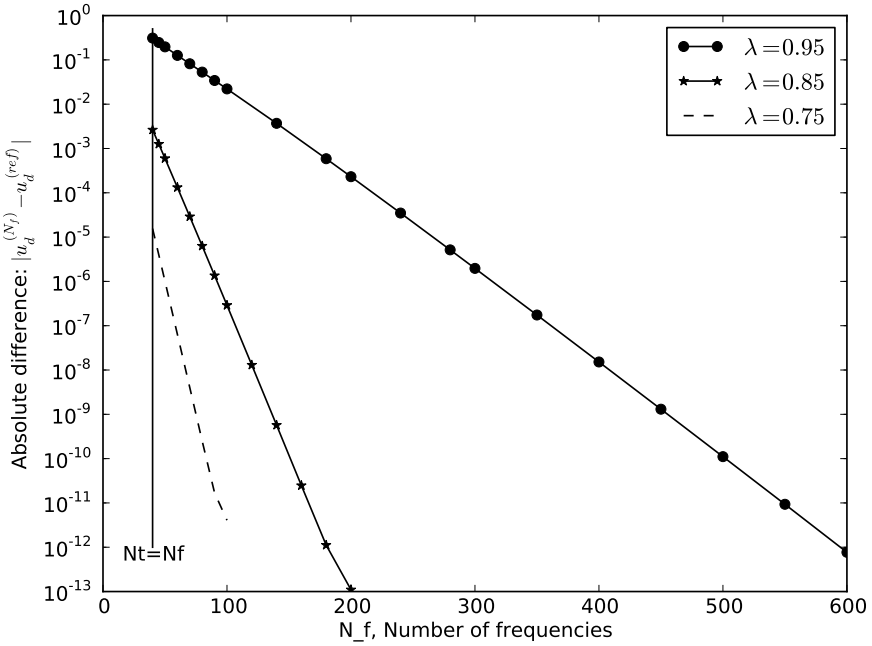

Figure 2: Absolute difference of the solution for the scattering of the unit sphere using indirect second kind formulation for three different λ. The accuracy obtained usually is indicated by $N_t = N_f$.

References

[1] LUBICH, C., *Convolution quadrature and discretized operational calculus. I*, Numerische Mathematik, (1988).

[2] LUBICH, C., *Convolution quadrature and discretized operational calculus. II*, Numerische Mathematik, (1988).

[3] HEROUX, M. AND AL., *An overview of the Trilinos project*, ACM Trans. Math. Software, 2005.

[4] REINDERS, J. , *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*, O'Reilly Media, 2007.

[5] BANJAI, L. AND SAUTER, S., *Rapid Solution of the Wave Equation in Unbounded Domains*, SIAM J. Numerical Analysis, 2008.

[6] BASTIAN, P. AND AL., *A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework*, Computing, 2008.

[7] BASTIAN, P. AND AL., *A generic grid interface for parallel and adaptive scienti?c computing. part II: Implementation and tests in DUNE*, Computing, 2008.

[8] BANJAI, L., *Multistep and Multistage Convolution Quadrature for the Wave Equation: Algorithms and Experiments*, SIAM SISC, (2010).

[9] SAUTER, S. A. AND SCHWAB, C., *Boundary element methods*, Springer, 2011.

[10] BEBENDORF, M., *Another software library on hierarchical matrices for elliptic differential equations*, 2012, `http://bebendorf.ins.uni-bonn.de/AHMED.html`.

[11] ŚMIGAJ W. AND AL. , *Solving Boundary Integral Problems with BEM++*, ACM Trans. Math. Software, to appear.

[12] BETCKE, T., SALLES N., AND SMIGAJ, W., *Exponentially accurate evaluation of time-stepping schemes for the wave equation via Convolution Quadrature type Methods*, To appear.