FMMLIB2D

Version 1.2

FORTRAN 90/95, MATLAB

April, 2012

User's guide

Contents

1 Introduction

This manual describes the use of the FMMLIB2D suite for the evaluation of potential fields, governed by either the Laplace or Helmholtz equation in free space. The codes are easy to use and reasonably well optimized for performance on either single core processors, or small multi-core systems using OpenMP. FMMLIB2D is being released under the terms of the GNU General Public License (version 2), as published by the Free Software Foundation.

The fast multipole method (FMM) computes N-body interactions in approximately linear time for non-pathological particle distributions, assuming in the case of the Helmholtz equation that the entire computational domain is a modest number of wavelengths in size. This is the "low frequency" regime from the point of view of either scattering theory or FMM implementations. (The high-frequency version of the FMM requires a more complex algorithm, and has not been incorporated into this software.) More precisely, FMMLIB2D computes sums of the form:

$$\phi(\mathbf{y}_i) = \sum_{j=1}^{N} q_j \, G_k(\mathbf{y}_i - \mathbf{x}_j) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} G_k(\mathbf{y}_i - \mathbf{x}_j) \tag{1}$$

for $i = 1, \ldots, N$, where

$$G_k(\mathbf{x}) = \frac{i}{4} H_0^{(1)}(k\|\mathbf{x}\|)$$

q_j is referred to as the charge strength and p_j as the dipole strength. $\mathbf{n} = (n_1, n_2)$ is a vector whose direction determines the dipole orientation (if present). For $k \neq 0$, we assume that k is in the upper half of the complex plane. It is designed for scattering calculations, and there are scaling issues that need to be incorporated to handle the modified Helmholtz (Yukawa) regime where k is both large and near the imaginary axis. This will be fixed in a forthcoming code release. When $k = 0$, we use $G_0(\mathbf{x}) = \log(\|\mathbf{x}\|)$. (The true Green's function requires a scaling by the factor $-\frac{1}{2\pi}$). There are several different routines available when $k = 0$. Subroutines with the prefix `lfmm2d` compute complex-valued sums of the form:

$$\phi(\mathbf{y}_i) = \sum_{j=1}^{N} q_j \, \log(\|\mathbf{y}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} \left(\log(\|\mathbf{y}_i - \mathbf{x}_j\|)\right). \tag{2}$$

Subroutines with the prefix `zfmm2d` compute complex-valued sums of the form:

$$\phi(\xi_i) = \sum_{j=1}^{N} p_j \, \frac{1}{\xi_i - z_j}. \tag{3}$$

Subroutines with the prefix `cfmm2d` compute complex-valued sums of the form:

$$\phi(\xi_i) = \sum_{j=1}^{N} q_j \, \log(\xi_i - z_j) + p_j \, \frac{1}{\xi_i - z_j}. \tag{4}$$

The `cfmm2d` routines are not intended for novice users, since the complex valued logarithm is a multi-valued function. As a result, the sums (4) have to be interpreted carefully and the routines are intended for advanced users only.

Important note: The charge and dipole strengths are assumed to be **complex** double precision numbers for both the Laplace and Helmholtz libraries. If you pass a **real** array, the code will not execute correctly.

This package provides a fully adaptive version of the FMM for the research community. It is not the most highly optimized version possible, intended rather to be accessible and modifiable with only modest effort. For a fully optimized code, far field and plane wave-based operators should be used [1]. This, however, would add significant complexity to the code, and would make the algorithm less transparent to the user and harder to modify. The internal documentation of lower level routines is mixed, but this is a work in progress. The higher level routines (we hope) should be clear.

In the next sections, we describe the calling sequences for the Fortran routines. The corresponding MATLAB routines are described in `Contents.m` in the `matlab` subdirectory.

1.1 Subroutine LFMM2DPARTSELF

---

subroutine lfmm2dpartself(ier, iprec, nsource, source, ifcharge, charge, ifdipole, dipstr, dipvec, ifpot, pot, ifgrad, grad, ifhess, hess)

computes sums of the form

$$\phi(\mathbf{x}_i) = \sum_{\substack{j=1 \\ j\neq i}}^{N} q_j \, \log(\|\mathbf{x}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} \left(\log(\|\mathbf{x}_i - \mathbf{x}_j\|)\right) \tag{5}$$

for $i = 1, \ldots, N$, as well as first and second derivatives of ϕ.

Input Parameters:

`iprec` ***integer*** :
precision flag. Allowed values are

`iprec` $= -2$	for least squares errors $< 0.5\,10^{0}$,
`iprec` $= -1$	for least squares errors $< 0.5\,10^{-1}$,
`iprec` $= 0$	for least squares errors $< 0.5\,10^{-2}$,
`iprec` $= 1$	for least squares errors $< 0.5\,10^{-3}$.
`iprec` $= 2$	for least squares errors $< 0.5\,10^{-6}$.
`iprec` $= 3$	for least squares errors $< 0.5\,10^{-9}$.
`iprec` $= 4$	for least squares errors $< 0.5\,10^{-12}$.
`iprec` $= 5$	for least squares errors $< 0.5\,10^{-14}$.

`nsource` ***integer*** :
number of sources

`source(2,nsource)` ***real *8*** :
sources(k,j) is the kth component of the jth source in $\mathbf{R}^2$.

`ifcharge` ***integer*** :
charge flag. If `icharge` $= 1$, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (5)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` $= 1$, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (5)).

`dipvec(2,nsource)` ***real *8*** :
dipvec(k,j) is the kth component of the orientation vector of the jth dipole ($\mathbf{n}_j$ in the formula (5)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the hessian of the potential is computed. Otherwise, it is not.

Unused arrays do not need to be allocated in full. Thus, if ifcharge = 0, charge can be dimensioned as a (complex) scalar. If ifdipole = 0, dipstr can be dimensioned as a complex scalar and dipvec can be dimensioned in the calling program as dipvec(2) - BUT NOT dipvec(1).

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(2,nsource)` ***complex *16*** :
grad(k,i) is the kth component of the gradient of the potential at the ith source

`hess(3,nsource)` ***complex *16*** :
hess(1,i), hess(2,i), and hess(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith source

---

Note that the charge, dipstr, pot, grad, hess arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.2 Subroutine LFMM2DPARTTARG

---

subroutine lfmm2dparttarg(ier, iprec, nsource, source, ifcharge, charge, ifdipole, dipstr, dipvec, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\mathbf{y}_i) = \sum_{j=1}^{N} q_j \, \log(\|\mathbf{y}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} \left(\log(\|\mathbf{y}_i - \mathbf{x}_j\|)\right)$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (5) if desired.

Input Parameters:

`iprec` ***integer*** :
precision flag. Allowed values are

`iprec` $= -2$	for least squares errors $< 0.5\,10^{0}$,
`iprec` $= -1$	for least squares errors $< 0.5\,10^{-1}$,
`iprec` $= 0$	for least squares errors $< 0.5\,10^{-2}$,
`iprec` $= 1$	for least squares errors $< 0.5\,10^{-3}$.
`iprec` $= 2$	for least squares errors $< 0.5\,10^{-6}$.
`iprec` $= 3$	for least squares errors $< 0.5\,10^{-9}$.
`iprec` $= 4$	for least squares errors $< 0.5\,10^{-12}$.
`iprec` $= 5$	for least squares errors $< 0.5\,10^{-15}$.

`nsource` ***integer*** :
number of sources

`source(2,nsource)` ***real *8*** :
sources(k,j) is the kth component of the jth source in $\mathbf{R}^2$.

`ifcharge` ***integer*** :
charge flag. If `icharge` $= 1$, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (5)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` $= 1$, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (5)).

`dipvec(2,nsource)` ***real *8*** :
dipvec(k,j) is the kth component of the orientation vector of the jth dipole ($\mathbf{n}_j$ in the formula (5)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the hessian of the potential is computed. Otherwise, it is not.

`ntarget` ***integer*** :
number of targets

`target(2,ntarget)` ***real *8*** :
target(k,j) is the kth component of the jth target in $\mathbf{R}^2$.

`ifpottarg` ***integer*** :
target potential flag. If `ifpottarg` = 1, the potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target gradient flag. If `ifgradtarg` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target hessian flag. If `ifhesstarg` = 1 the hessian of the potential is computed. Otherwise, it is not.

Unused arrays do not need to be allocated in full. Thus, if ifcharge = 0, charge can be dimensioned as a (complex) scalar. If ifdipole = 0, dipstr can be dimensioned as a complex scalar and dipvec can be dimensioned in the calling program as dipvec(2) - BUT NOT dipvec(1).

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(2,nsource)` ***complex *16*** :
grad(k,i) is the kth component of the field (-gradient of the potential) at the ith source

`hess(3,nsource)` ***complex *16*** :
hess(1,i), hess(2,i), and hess(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(2,ntarget)` ***complex *16*** :
gradtarg(k,i) is the kth component of the field (-gradient of the potential) at the ith target

`hesstarg(3,ntarget)` ***complex *16*** :
hesstarg(1,i), hesstarg(2,i), and hesstarg(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith target

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.3 Subroutine L2DPARTDIRECT

---

subroutine l2dpartdirect(nsource, source, ifcharge, charge, ifdipole, dipstr, dipvec, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\mathbf{y}_i) = \sum_{j=1}^{N} q_j \log(\|\mathbf{y}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} (\log(\|\mathbf{y}_i - \mathbf{x}_j\|))$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (5) if desired. **It implements the summation formula directly and is not fast.**

Input Parameters:

`nsource` ***integer*** :
number of sources

`source(2,nsource)` ***real *8*** :
sources(k,j) is the kth component of the jth source in $\mathbf{R}^2$.

`ifcharge` ***integer*** :
charge flag. If `icharge` = 1, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (5)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` = 1, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (5)).

`dipvec(2,nsource)` ***real *8*** :
dipvec(k,j) is the kth component of the orientation vector of the jth dipole ($\mathbf{n}_j$ in the formula (5)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the hessian of the potential is computed. Otherwise, it is not.

`ntarget` ***integer*** :
number of targets

`target(2,ntarget)` ***real *8*** :
target(k,j) is the kth component of the jth target in $\mathbf{R}^2$.

`ifpottarg` ***integer*** :
target potential flag. If `ifpottarg` = 1, the potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target gradient flag. If `ifgradtarg` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target hessian flag. If `ifhesstarg` = 1 the hessian of the potential is computed. Otherwise, it is not.

Unused arrays do not need to be allocated in full. Thus, if ifcharge = 0, charge can be dimensioned as a (complex) scalar. If ifdipole = 0, dipstr can be dimensioned as a complex scalar and dipvec can be dimensioned in the calling program as dipvec(2) - BUT NOT dipvec(1).

Output Parameters:

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(2,nsource)` ***complex *16*** :
grad(k,i) is the kth component of the field (-gradient of the potential) at the ith source

`hess(3,nsource)` ***complex *16*** :
hess(1,i), hess(2,i), and hess(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(2,ntarget)` ***complex *16*** :
gradtarg(k,i) is the kth component of the field (-gradient of the potential) at the ith target

`hesstarg(3,ntarget)` ***complex *16*** :
hesstarg(1,i), hesstarg(2,i), and hesstarg(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith target

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.4 Subroutine HFMM2DPARTSELF

---

subroutine hfmm2dpartself(ier, iprec, zk, nsource, source, ifcharge, charge, ifdipole, dipstr, dipvec, ifpot, pot, ifgrad, grad, ifhess, hess)

computes sums of the form

$$\phi(\mathbf{x}_i) = \frac{i}{4} \sum_{\substack{j=1 \\ j \neq i}}^{N} q_j \, H_0^{(1)}(k\|\mathbf{x}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} \left(H_0^{(1)}(k\|\mathbf{x}_i - \mathbf{x}_j\|) \right) \tag{6}$$

for $i = 1, \ldots, N$, as well as first and second derivatives of ϕ.

Input Parameters:

`iprec` ***integer*** :
precision flag. Allowed values are

`iprec` $= -2$	for least squares errors $< 0.5\,10^{0}$,
`iprec` $= -1$	for least squares errors $< 0.5\,10^{-1}$,
`iprec` $= 0$	for least squares errors $< 0.5\,10^{-2}$,
`iprec` $= 1$	for least squares errors $< 0.5\,10^{-3}$.
`iprec` $= 2$	for least squares errors $< 0.5\,10^{-6}$.
`iprec` $= 3$	for least squares errors $< 0.5\,10^{-9}$.
`iprec` $= 4$	for least squares errors $< 0.5\,10^{-12}$.
`iprec` $= 5$	for least squares errors $< 0.5\,10^{-14}$.

`zk` ***complex *16*** :
Helmholtz parameter

`nsource` ***integer*** :
number of sources

`source(2,nsource)` ***real *8*** :
sources(k,j) is the kth component of the jth source in $\mathbf{R}^2$.

`ifcharge` ***integer*** :
charge flag. If `icharge` $= 1$, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (6)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` $= 1$, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (6)).

`dipvec(2,nsource)` ***real *8*** :
dipvec(k,j) is the kth component of the orientation vector of the jth dipole ($\mathbf{n}_j$ in the formula (6)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the hessian of the potential is computed. Otherwise, it is not.

Unused arrays do not need to be allocated in full. Thus, if ifcharge = 0, charge can be dimensioned as a (complex) scalar. If ifdipole = 0, dipstr can be dimensioned as a complex scalar and dipvec can be dimensioned in the calling program as dipvec(2) - BUT NOT dipvec(1).

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(2,nsource)` ***complex *16*** :
grad(k,i) is the kth component of the gradient of the potential at the ith source

`hess(3,nsource)` ***complex *16*** :
hess(1,i), hess(2,i), and hess(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith source

---

Note that the charge, dipstr, pot, grad, hess arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.5 Subroutine HFMM2DPARTTARG

---

subroutine hfmm2dparttarg(ier, iprec, zk, nsource, source, ifcharge, charge, ifdipole, dipstr, dipvec, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\mathbf{y}_i) = \frac{i}{4}\sum_{j=1}^{N} q_j\, H_0^{(1)}(k\|\mathbf{y}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j}\left(H_0^{(1)}(k\|\mathbf{y}_i - \mathbf{x}_j\|)\right)$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (6) if desired.

Input Parameters:

`iprec` ***integer*** :
precision flag. Allowed values are

`iprec` $= -2$	for least squares errors $< 0.5\,10^{0}$,
`iprec` $= -1$	for least squares errors $< 0.5\,10^{-1}$,
`iprec` $= 0$	for least squares errors $< 0.5\,10^{-2}$,
`iprec` $= 1$	for least squares errors $< 0.5\,10^{-3}$.
`iprec` $= 2$	for least squares errors $< 0.5\,10^{-6}$.
`iprec` $= 3$	for least squares errors $< 0.5\,10^{-9}$.
`iprec` $= 4$	for least squares errors $< 0.5\,10^{-12}$.
`iprec` $= 5$	for least squares errors $< 0.5\,10^{-15}$.

`zk` ***complex *16*** :
Helmholtz parameter

`nsource` ***integer*** :
number of sources

`source(2,nsource)` ***real *8*** :
sources(k,j) is the kth component of the jth source in $\mathbf{R}^2$.

`ifcharge` ***integer*** :
charge flag. If `icharge` $= 1$, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (6)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` $= 1$, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (6)).

`dipvec(2,nsource)` ***real *8*** :
dipvec(k,j) is the kth component of the orientation vector of the jth dipole ($\mathbf{n}_j$ in the formula (6)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the hessian of the potential is computed. Otherwise, it is not.

`ntarget` ***integer*** :
number of targets

`target(2,ntarget)` ***real *8*** :
target(k,j) is the kth component of the jth target in $\mathbf{R}^2$.

`ifpottarg` ***integer*** :
target potential flag. If `ifpottarg` = 1, the potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target gradient flag. If `ifgradtarg` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target hessian flag. If `ifhesstarg` = 1 the hessian of the potential is computed. Otherwise, it is not.

Unused arrays do not need to be allocated in full. Thus, if ifcharge = 0, charge can be dimensioned as a (complex) scalar. If ifdipole = 0, dipstr can be dimensioned as a complex scalar and dipvec can be dimensioned in the calling program as dipvec(2) - BUT NOT dipvec(1).

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(2,nsource)` ***complex *16*** :
grad(k,i) is the kth component of the field (-gradient of the potential) at the ith source

`hess(3,nsource)` ***complex *16*** :
hess(1,i), hess(2,i), and hess(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(2,ntarget)` ***complex *16*** :
gradtarg(k,i) is the kth component of the field (-gradient of the potential) at the ith target

`hesstarg(3,ntarget)` ***complex *16*** :
hesstarg(1,i), hesstarg(2,i), and hesstarg(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith target

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.6 Subroutine H2DPARTDIRECT

---

subroutine h2dpartdirect(zk, nsource, source, ifcharge, charge, ifdipole, dipstr, dipvec, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\mathbf{y}_i) = \frac{i}{4} \sum_{j=1}^{N} q_j \, H_0^{(1)}(k\|\mathbf{y}_i - \mathbf{x}_j\|) + p_j \mathbf{n}_j \cdot \nabla_{\mathbf{x}_j} \left(H_0^{(1)}(k\|\mathbf{y}_i - \mathbf{x}_j\|) \right)$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (6) if desired. **It implements the summation formula directly and is not fast.**

Input Parameters:

`zk` ***complex *16*** :
Helmholtz parameter

`nsource` ***integer*** :
number of sources

`source(2,nsource)` ***real *8*** :
sources(k,j) is the kth component of the jth source in $\mathbf{R}^2$.

`ifcharge` ***integer*** :
charge flag. If `icharge` = 1, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (6)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` = 1, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (6)).

`dipvec(2,nsource)` ***real *8*** :
dipvec(k,j) is the kth component of the orientation vector of the jth dipole ($\mathbf{n}_j$ in the formula (6)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the hessian of the potential is computed. Otherwise, it is not.

`ntarget` ***integer*** :
number of targets

`target(2,ntarget)` ***real *8*** :
target(k,j) is the kth component of the jth target in $\mathbf{R}^2$.

`ifpottarg` ***integer*** :
target potential flag. If `ifpottarg` = 1, the potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target gradient flag. If `ifgradtarg` = 1 the gradient of the potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target hessian flag. If `ifhesstarg` = 1 the hessian of the potential is computed. Otherwise, it is not.

Unused arrays do not need to be allocated in full. Thus, if ifcharge = 0, charge can be dimensioned as a (complex) scalar. If ifdipole = 0, dipstr can be dimensioned as a complex scalar and dipvec can be dimensioned in the calling program as dipvec(2) - BUT NOT dipvec(1).

Output Parameters:

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(2,nsource)` ***complex *16*** :
grad(k,i) is the kth component of the field (-gradient of the potential) at the ith source

`hess(3,nsource)` ***complex *16*** :
hess(1,i), hess(2,i), and hess(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(2,ntarget)` ***complex *16*** :
gradtarg(k,i) is the kth component of the field (-gradient of the potential) at the ith target

`hesstarg(3,ntarget)` ***complex *16*** :
hesstarg(1,i), hesstarg(2,i), and hesstarg(3,i) are ∂_{xx}, ∂_{xy}, and ∂_{yy} derivatives of the potential at the ith target

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.7 Subroutine ZFMM2DPARTSELF

---

subroutine zfmm2dpartself(ier, iprec, nsource, source, dipstr, ifpot, pot, ifgrad, grad, ifhess, hess)

computes sums of the form

$$\phi(z_i) = \sum_{\substack{j=1 \\ j \neq i}}^{N} p_j \frac{1}{z_i - z_j} \tag{7}$$

for $i = 1, \ldots, N$, as well as first and second derivatives of ϕ.

Input Parameters:

`iprec` ***integer*** :
precision flag. Allowed values are

`iprec` $= -2$	for least squares errors $< 0.5\,10^{0}$,
`iprec` $= -1$	for least squares errors $< 0.5\,10^{-1}$,
`iprec` $= 0$	for least squares errors $< 0.5\,10^{-2}$,
`iprec` $= 1$	for least squares errors $< 0.5\,10^{-3}$.
`iprec` $= 2$	for least squares errors $< 0.5\,10^{-6}$.
`iprec` $= 3$	for least squares errors $< 0.5\,10^{-9}$.
`iprec` $= 4$	for least squares errors $< 0.5\,10^{-12}$.
`iprec` $= 5$	for least squares errors $< 0.5\,10^{-14}$.

`nsource` ***integer*** :
number of sources

`source(nsource)` ***complex *16*** :
sources(j) is the location of the jth source in the complex plane.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (7)).

`ifpot` ***integer*** :
potential flag. If `ifpot` $= 1$, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` $= 1$ the first derivative of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` $= 1$ the second derivative of the potential is computed. Otherwise, it is not.

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(nsource)` ***complex *16*** :
grad(i) is the first derivative of the potential at the ith source

`hess(nsource)` ***complex *16*** :
hess(i) is the second derivative of the potential at the ith source

---

Note that the dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the dipole strengths are real).

1.8 Subroutine ZFMM2DPARTTARG

---

subroutine zfmm2dparttarg(ier, iprec, nsource, source, dipstr, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\xi_i) = \sum_{j=1}^{N} p_j \frac{1}{\xi_i - z_j}$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (7) if desired.

Input Parameters:

`nsource` ***integer*** :
number of sources

`source(nsource)` ***complex *16*** :
sources(j) is the location of the jth source in the complex plane.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (7)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the first derivative of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the second derivative of the potential is computed. Otherwise, it is not.

`target(ntarget)` ***complex *16*** :
target(j) is the location of the jth target in the complex plane.

`ifpottarg` ***integer*** :
target potential flag. If `ifpot` = 1, the target potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target derivative flag. If `ifgrad` = 1 the first derivative of the target potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target second derivative flag. If `ifhess` = 1 the second derivative of the target potential is computed. Otherwise, it is not.

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(nsource)` ***complex *16*** :
grad(i) is the first derivative of the potential at the ith source

`hess(nsource)` ***complex *16*** :
hess(i) is the second derivative of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(ntarget)` ***complex *16*** :
gradtarg(i) is the first derivative of the potential at the ith target

`hesstarg(ntarget)` ***complex *16*** :
hesstarg(i) is the second derivative of the potential at the ith target

---

Note that the dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the dipole strengths are real).

1.9 Subroutine Z2DPARTDIRECT

---

subroutine z2dpartdirect(nsource, source, dipstr, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\xi_i) = \sum_{j=1}^{N} p_j \frac{1}{\xi_i - z_j}$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (7) if desired. **It implements the summation formula directly and is not fast.**

Input Parameters:

`nsource` ***integer*** :
number of sources

`source(nsource)` ***complex *16*** :
sources(j) is the location of the jth source in the complex plane.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (7)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
derivative flag. If `ifgrad` = 1 the first derivative of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
second derivative flag. If `ifhess` = 1 the second derivative of the potential is computed. Otherwise, it is not.

`target(ntarget)` ***complex *16*** :
target(j) is the location of the jth target in the complex plane.

`ifpottarg` ***integer*** :
target potential flag. If `ifpot` = 1, the target potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target derivative flag. If `ifgrad` = 1 the first derivative of the target potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target second derivative flag. If `ifhess` = 1 the second derivative of the target potential is computed. Otherwise, it is not.

Output Parameters:

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(nsource)` ***complex *16*** :
grad(i) is the first derivative of the potential at the ith source

`hess(nsource)` ***complex *16*** :
hess(i) is the second derivative of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(ntarget)` ***complex *16*** :
gradtarg(i) is the first derivative of the potential at the ith target

`hesstarg(ntarget)` ***complex *16*** :
hesstarg(i) is the second derivative of the potential at the ith target

---

Note that the dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the dipole strengths are real).

1.10 Subroutine CFMM2DPARTSELF

---

subroutine cfmm2dpartself(ier, iprec, nsource, source, ifcharge, charge, ifdipole, dipstr, ifpot, pot, ifgrad, grad, ifhess, hess)

computes sums of the form

$$\phi(z_i) = \sum_{\substack{j=1 \\ j\neq i}}^{N} q_j \, \log(z_i - z_j) + p_j \, \frac{1}{z_i - z_j} \tag{8}$$

for $i = 1, \ldots, N$, as well as first and second derivatives of ϕ. Note, that this is a highly specialized low level routine, which should be used only by advanced users. The complex valued logarithm is a multivalued function, therefore, the potential values returned by cfmm2d have to be interpreted carefully, if charges are specified. For example, only the real part of the potential is meaningful for the real valued charges. The derivatives are single-valued and are valid for arbitrary valued complex charges and dipoles.

Input Parameters:

`iprec` ***integer*** :
precision flag. Allowed values are

`iprec` $= -2$	for least squares errors $< 0.5\,10^{0}$,
`iprec` $= -1$	for least squares errors $< 0.5\,10^{-1}$,
`iprec` $= 0$	for least squares errors $< 0.5\,10^{-2}$,
`iprec` $= 1$	for least squares errors $< 0.5\,10^{-3}$.
`iprec` $= 2$	for least squares errors $< 0.5\,10^{-6}$.
`iprec` $= 3$	for least squares errors $< 0.5\,10^{-9}$.
`iprec` $= 4$	for least squares errors $< 0.5\,10^{-12}$.
`iprec` $= 5$	for least squares errors $< 0.5\,10^{-14}$.

`nsource` ***integer*** :
number of sources

`source(nsource)` ***complex *16*** :
sources(j) is the location of the jth source in the complex plane.

`ifcharge` ***integer*** :
charge flag. If `icharge` $= 1$, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (8)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` $= 1$, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (8)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the first derivative of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the second derivative of the potential is computed. Otherwise, it is not.

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(nsource)` ***complex *16*** :
grad(i) is the first derivative of the potential at the ith source

`hess(nsource)` ***complex *16*** :
hess(i) is the second derivative of the potential at the ith source

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.11 Subroutine CFMM2DPARTTARG

---

subroutine cfmm2dparttarg(ier, iprec, nsource, source, ifcharge, charge, ifdipole, dipstr, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\xi_i) = \sum_{j=1}^{N} q_j \, \log(\xi_i - z_j) + p_j \, \frac{1}{\xi_i - z_j}$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (8) if desired. Note, that this is a highly specialized low level routine, which should be used only by advanced users. The complex valued logarithm is a multivalued function, therefore, the potential values returned by cfmm2d have to be interpreted carefully, if charges are specified. For example, only the real part of the potential is meaningful for the real valued charges. The derivatives are single-valued and are valid for arbitrary valued complex charges and dipoles.

Input Parameters:

`nsource` ***integer*** :
number of sources

`source(nsource)` ***complex *16*** :
sources(j) is the location of the jth source in the complex plane.

`ifcharge` ***integer*** :
charge flag. If `icharge` = 1, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (8)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` = 1, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (8)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
gradient flag. If `ifgrad` = 1 the first derivative of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
hessian flag. If `ifhess` = 1 the second derivative of the potential is computed. Otherwise, it is not.

`target(ntarget)` ***complex *16*** :
target(j) is the location of the jth target in the complex plane.

`ifpottarg` ***integer*** :
target potential flag. If `ifpot` = 1, the target potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target derivative flag. If `ifgrad` = 1 the first derivative of the target potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target second derivative flag. If `ifhess` = 1 the second derivative of the target potential is computed. Otherwise, it is not.

Output Parameters:

`ier` ***integer*** :
Error return codes.
`ier = 0`: Successful completion of code.
`ier = 4`: failure to allocate memory for oct-tree
`ier = 8`: failure to allocate memory for FMM workspaces
`ier = 16`: failure to allocate meory for multipole/local expansions

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(nsource)` ***complex *16*** :
grad(i) is the first derivative of the potential at the ith source

`hess(nsource)` ***complex *16*** :
hess(i) is the second derivative of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(ntarget)` ***complex *16*** :
gradtarg(i) is the first derivative of the potential at the ith target

`hesstarg(ntarget)` ***complex *16*** :
hesstarg(i) is the second derivative of the potential at the ith target

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

1.12 Subroutine C2DPARTDIRECT

---

subroutine c2dpartdirect(nsource, source, ifcharge, charge, ifdipole, dipstr, ifpot, pot, ifgrad, grad, ifhess, hess, ntarget, target, ifpottarg, pottarg, ifgradtarg, gradtarg, ifhesstarg, hesstarg)

compute sums of the form

$$\phi(\xi_i) = \sum_{j=1}^{N} q_j \, \log(\xi_i - z_j) + p_j \, \frac{1}{\xi_i - z_j}$$

for $i = 1, \ldots, N_t$, as well as first and second derivatives of ϕ. It also returns sums of the form (8) if desired. **It implements the summation formula directly and is not fast.** Note, that this is a highly specialized low level routine, which should be used only by advanced users. The complex valued logarithm is a multivalued function, therefore, the potential values returned by cfmm2d have to be interpreted carefully, if charges are specified. For example, only the real part of the potential is meaningful for the real valued charges. The derivatives are single-valued and are valid for arbitrary valued complex charges and dipoles.

Input Parameters:

`nsource` ***integer*** :
number of sources

`source(nsource)` ***complex *16*** :
sources(j) is the location of the jth source in the complex plane.

`ifcharge` ***integer*** :
charge flag. If `icharge` = 1, then include the effect of the charge sources. Otherwise, omit.

`charge(nsource)` ***complex *16*** :
charge(j) is the strength of the jth charge (q_j in the formula (8)).

`ifdipole` ***integer*** :
dipole flag. If `idipole` = 1, then include the effect of the dipole sources. Otherwise, omit.

`dipstr(nsource)` ***complex *16*** :
dipstr(j) is the strength of the jth dipole (p_j in the formula (8)).

`ifpot` ***integer*** :
potential flag. If `ifpot` = 1, the potential is computed. Otherwise, it is not.

`ifgrad` ***integer*** :
derivative flag. If `ifgrad` = 1 the first derivative of the potential is computed. Otherwise, it is not.

`ifhess` ***integer*** :
second derivative flag. If `ifhess` = 1 the second derivative of the potential is computed. Otherwise, it is not.

`target(ntarget)` ***complex *16*** :
target(j) is the location of the jth target in the complex plane.

`ifpottarg` ***integer*** :
target potential flag. If `ifpot` = 1, the target potential is computed. Otherwise, it is not.

`ifgradtarg` ***integer*** :
target derivative flag. If `ifgrad` = 1 the first derivative of the target potential is computed. Otherwise, it is not.

`ifhesstarg` ***integer*** :
target second derivative flag. If `ifhess` = 1 the second derivative of the target potential is computed. Otherwise, it is not.

Output Parameters:

`pot(nsource)` ***complex *16*** :
pot(i) is the potential at the ith source

`grad(nsource)` ***complex *16*** :
grad(i) is the first derivative of the potential at the ith source

`hess(nsource)` ***complex *16*** :
hess(i) is the second derivative of the potential at the ith source

`pottarg(ntarget)` ***complex *16*** :
pottarg(i) is the potential at the ith target

`gradtarg(ntarget)` ***complex *16*** :
gradtarg(i) is the first derivative of the potential at the ith target

`hesstarg(ntarget)` ***complex *16*** :
hesstarg(i) is the second derivative of the potential at the ith target

---

Note that the charge, dipstr, pot, grad, hess, pottarg, gradtarg, hesstarg, arrays must be declared and passed as complex arrays (even if the charge and dipole strengths are real).

---

2 Sample drivers for FMMLIB2D

In the FMM2D/examples directory, the file `lfmm2dpart_driver.f` contains a sample driver for `lfmm2dparttarg`. It creates a random distribution of source points on the unit circle centered at the origin a random distribution of target points on a separated unit circle, centered at $(3, 0)$. The code then computes the potential and field at all source and target points. On a single core, with 100,000 sources, 100,000 targets, and iprec=4, the execution time should be one or two seconds.

The file `hfmm2dpart_driver.f` contains a sample driver for `hfmm2dparttarg`. It creates the same distribution of sources and targets, and sets the Helmholtz parameter to $k = 20$. On a single core, with 100,000 sources, 100,000 targets, and iprec=4, the execution time should be about four seconds.

Sample drivers for the MATLAB routines can be found in the `matlab` directory in files `test_lfmm2dpart_direct.m`, `test_hfmm2dpart_direct.m`.

3 Acknowledgments

This work was supported by the Department of Energy under contract DE-FGO288-ER-25053, by the Air Force Office of Scientific Research under MURI grant FA9550-06-1-0337 and NSSEFF Program Award FA9550-10-1-0180, by the National Science Foundation under grant DMS09-34733, and by a research grant from Meyer Sound Laboratories, Inc.

References

[1] W. Crutchfield, Z. Gimbutas, L. Greengard, J. Huang, V. Rokhlin, N. Yarvin, and J. Zhao, "Remarks on the implementation of the wideband FMM for the Helmholtz equation in two dimensions," Contemporary Mathematics **408**, pp. 99-110 (2006).

[2] J. Carrier, L. Greengard, and V. Rokhlin, "A Fast Adaptive Multipole Algorithm for Particle Simulations," SIAM J. Sci. and Stat. Comput. **9**, pp. 669–686 (1988).