A Matlab-Based Finite Difference Solver for the Poisson Problem with Mixed Dirichlet-Neumann Boundary Conditions

Ashton S. Reimer[a)] Alexei F. Cheviakov[b)]
Department of Mathematics and Statistics, University of Saskatchewan, Saskatoon, S7N 5E6 Canada

April 17, 2012

Abstract

A Matlab-based finite-difference numerical solver for the Poisson equation for a rectangle and a disk in two dimensions, and a spherical domain in three dimensions, is presented. The solver is optimized for handling an arbitrary combination of Dirichlet and Neumann boundary conditions, and allows for full user control of mesh refinement. The solver routines utilize effective and parallelized sparse vector and matrix operations. Computations exhibit high speeds, numerical stability with respect to mesh size and mesh refinement, and acceptable error values even on desktop computers.

PROGRAM SUMMARY/NEW VERSION PROGRAM SUMMARY

Manuscript Title: A Matlab-based Finite-difference Solver for the Poisson Problem in Two and Three Dimensions, Optimized for Strongly Heterogeneous Boundary Conditions
Authors: Ashton S. Reimer and Alexei F. Cheviakov
Program Title: FDMRP 1.0 - Finite-Difference with Mesh Refinement Poisson equation solver
Journal Reference:
Catalogue identifier:
Licensing provisions: GNU General Public License v3.0
Programming language: Matlab 2010a
Computer: PC, Macintosh
Operating system: Windows, OSX, Linux
RAM: 8GB (8,589,934,592 bytes)
Number of processors used: Single Processor, Dual Core
Keywords: Poisson problem, Finite-difference solver, Matlab, Strongly heterogeneous boundary conditions, Narrow Escape Problems
Classification: 4.3 Differential Equations
Nature of problem: To solve the Poisson problem in a standard domain with "patchy surface"-type (strongly heterogeneous) Neumann/Dirichlet boundary conditions.
Solution method: Finite difference with mesh refinement.
Restrictions: Spherical domain in 3D; rectangular domain or a disk in 2D.
Unusual features: Choice between mldivide/iterative solver for the solution of large system of linear algebraic equations that arise. Full user control of Neumann/Dirichlet boundary conditions and mesh refinement.
Running time: Depending on the number of points taken and the geometry of the domain, the routine may take from less than a second to several hours to execute.

1 Introduction

The Laplacian operator is a fundamental component in a large number of multi-dimensional linear models of mathematical physics; to name a few, the Laplacian arises in description of various

[a)] Electronic mail: ashton.reimer@usask.ca
[b)] Electronic mail: cheviakov@math.usask.ca

classical and quantum wave phenomena, as well as models involving diffusion and viscosity effects.

Linear time-independent boundary value problems (BVP) for Laplace and Poisson equations constitute a special class of important problems. The Laplace operator is separable in many classical and esoteric coordinate systems [7,8]. The corresponding linear problems can be solved analytically for specific domains which correspond to parallelepipeds in the respective coordinate systems, provided that boundary conditions are sufficiently simple. For a general BVP, however, one normally has to rely on numerical methods.

Finite-difference methods are common numerical methods for solution of linear second-order time-independent partial differential equations. Finite-difference methods involve discretization of the spatial domain, the differential equation, and boundary conditions, and a subsequent solution of a large system of linear equations for the approximate solution values in the nodes of the numerical mesh. Simplest discretizations of second-order differential operators commonly have first or second order accuracy. The resulting large system of linear equations involves a sparse matrix and are solved by iterative methods (Jacobi, Gauss-Seigel, etc.) or Gaussian elimination/LU decomposition, which have been significantly optimized for sparse matrices.

The current work is motivated by BVPs for the Poisson equation where boundary correspond to so called "patchy surfaces", i.e., are strongly heterogeneous, involving combination of Neumann and Dirichlet boundary conditions on different parts of the boundary. Models involving patchy surface BVPs are found in various fields. For example, models of vacuum tubes with patchy surfaces in electrostatics were considered in [5, 6]. Cell signalling models in biology are concerned with motion of ions that seek an open channel located in the cell membrane; the latter is modelled as a combination of homogeneous reflecting (Neumann) and absorbing (Dirichlet) boundary conditions (e.g., [1,10]). In particular, the continuum limit, the mean first passage time (MFPT) $v(x)$ required for a Brownian particle to leave a domain with a reflecting boundary and absorbing traps on the boundary satisfies the mixed Dirichlet-Neumann problem (cf. [3])

$$\begin{aligned} &\Delta v = -\frac{1}{D}, \quad x \in \Omega, \\ &v = 0, \quad x \in \partial\Omega_a = \bigcup_{j=1}^{N} \partial\Omega_{\epsilon_j}, \quad j = 1, ..., N; \quad \partial_n v = 0, \quad x \in \partial\Omega_r, \end{aligned} \tag{1.1}$$

where D is the diffusion coefficient and may be constant or a function of space. Dirichlet boundary parts Ω_{ϵ_j} correspond to small absorbing traps, and the boundary part Ω_r corresponds to reflective (Neumann) boundary conditions. In order to determine the applicability limits of the approximate solutions derived in [2] and [9], a numerical solution to the MFPT problem was required for comparison. The numerical solver described below was initially developed to solve MFPT problems (1.1).

The programs presented in the current work are rather general; they provide numerical solutions of Poisson boundary value problems

$$\begin{aligned} &\Delta v = f(x), \quad x \in \Omega, \\ &v = v_D(x), \quad x \in \partial\Omega_D, \quad \partial_n v = v_N(x), \quad x \in \partial\Omega_N, \end{aligned} \tag{1.2}$$

involving the combination of Dirichlet boundary conditions on the part Ω_D of the boundary, and Neumann boundary conditions on the part Ω_N of the boundary. The forcing term $f(x)$ is arbitrary; Ω is either a rectangle or a disk in two dimensions, or a sphere in three dimensions.

All programs were implemented in `Matlab`, and are respectively optimized to avoid loops and use vector and matrix operations even at the stage of problem formulation, which allowed for a significant speedup of computation. The programs are compatible with `Octave`, the open-source `Matlab` equivalent, with the exception that the iterative solving options are unavailable in `Octave`. While many general purpose finite-element software packages exist (e.g., `ELMER`, `COMSOL`, etc.), such

"black-box" packages require a significant investment of time and/or money, not offering sufficient insight into the numerical methods used and their implementation. The `Matlab`-based numerical solvers described in the current contribution offer a transparent, simple-to-use way to solve Poisson problems in simple geometries with a finite-difference method.

Normally, a second-order symmetric discretization of the Laplacian operator was used. The programs allow for mesh refinement through variable step sizes in any direction (in which case the scheme becomes generally first-order accurate). Mesh refinement allows one to concentrate the mesh around given point(s) within the computation domain Ω or on its boundary, where higher resolution is needed. For example, for MFPT problems with small boundary or volume traps, mesh refinement was used to resolve the trap regions accurately, and have coarser mesh far from traps where the solution changes slowly. The properties of the finite-difference approximation, including implications of singularities in curvilinear coordinates, are discussed in Section 2.

In a two- or three-dimensional domain, the discretization of the Poisson BVP (1.2) yields a system of sparse linear algebraic equations containing $\mathcal{N} = LM$ equations for two-dimensional domains, and $\mathcal{N} = LMN$ equations for three-dimensional domains, where L, M, N are the numbers of steps in the corresponding directions. In order to solve the large sparse system, the programs take advantage of highly effective `Matlab` routines: `mldivide` and `bicg`. For example, on a system with 8 gigabytes of memory, for the spherical domain, it was possible to use a total of $\mathcal{N} \sim 1.6 \cdot 10^7$ data points, using the values $L = 400, M = N = 200$ (in the directions of the polar angle, radius, and azimuthal angle, respectively). Depending on the number of points used and hardware configurations, the runtime can vary from fractions of a second to hours.

Run examples for problems in a rectangle, disk, and sphere with homogeneous and nonhomogeneous boundary conditions are presented in Sections 3–5. The examples include error plots for trial solutions which were known exactly. For all computed configurations where exact solutions are not known, stability of the numerical results with respect to mesh size and mesh refinement functions was experimentally verified.

The presented programs can be easily generalized to other sets of curvilinear coordinates using similar considerations and a corresponding different set of scaling (Lamé) factors.

2 The Finite-Difference Approximation

Finite-difference methods approximate the derivative of a function at a given point by a finite difference. In one dimension, $a \le x \le b$, consider a function $u(x)$ and a numerical mesh $a = x_1 < ... < x_n = b$. Let $u(x_i) \equiv u_i$. The mesh step sizes are given by $h_i = x_{i+1} - x_i$. A derivative of $u(x)$ at $x = x_i$ can be approximated by, for example, a forward difference

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_{i+1} - u_i}{h_i} \tag{2.1}$$

or a central difference

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_{i+1} - u_{i-1}}{h_i + h_{i-1}}, \tag{2.2}$$

and the second derivative by a central difference

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i = \left(\frac{2}{h_i + h_{i-1}}\right)\left(\frac{u_{i+1} - u_i}{h_i} - \frac{u_i - u_{i-1}}{h_{i-1}}\right). \tag{2.3}$$

Formulas (2.2) and (2.3) have second-order accuracy if all $h_i = h = \text{const}$, and are first-order accurate otherwise.

We now present the discretized versions of the Dirichlet and Neumann boundary conditions and the Poisson equations operator in Cartesian, polar and spherical coordinates.

2.1 Approximation of Boundary Conditions

For a Dirichlet boundary condition, e.g., $u(b) = A$, one one simply sets the approximate numerical solution at a given boundary point to equal the boundary value:

$$u_1 = A.$$

In order to apply Neumann boundary conditions of the form $\frac{\partial u}{\partial x}(a) = A$ in one dimension, the simplest choice is to use a first-order approximation given by a backward difference, and hence one has

$$u_1 = u_2 - h_1 A.$$

This formula naturally generalizes to multiple dimensions and curvilinear geometries, where a Neumann boundary condition corresponds to the value of the normal derivative prescribed at a given boundary point.

As it is well known, solutions to Poisson equation with a mixture of Dirichlet and Neumann boundary conditions can involve mild, square-root-type singularities, such that the solution derivative can become unbounded at a boundary point where the boundary condition changes from Dirichlet to Neumann (see, e.g., [4]). This however does not present any additional difficulties. For example, considering a Poisson problem in a square domain

$$\begin{aligned} &\Delta u = 4, \quad 0 \le x, y \le 1, \\ &u_x(0, y) = 0, \quad u(1, y) = 1 + y^2, \\ &u_y(x \le 0.5, 0) = 0, \quad u(x > 0.5, 0) = x^2, \\ &u_y(x \le 0.5, 1) = 2, \quad u(x > 0.5, 1) = x^2 + 1 \end{aligned} \tag{2.4}$$

with an exact solution

$$u(x, y) = x^2 + y^2,$$

one observes that near the point of the change from Neumann to Dirichlet boundary conditions at $x = 0.5$, the relative error is controlled for various mesh sizes (Fig. 1). [For run examples of the Poisson solver in rectangular domains, see Section 3.]

2.2 Approximation of the PDE

For the 2D Poisson equation in a rectangle,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \tag{2.5}$$

the approximate solution is given by $u_{i,j} \simeq u(x_i, y_j)$, $i = 1, \ldots, L$, $j = 1, \ldots, M$. The variable mesh step sizes in x and y directions are labeled by h_i^y and h_j^y, respectively. Using equation (2.3), one obtains

$$\begin{aligned} &\left(\frac{2}{h_i^x + h_{i-1}^x}\right)\left(\frac{u_{i+1,j} - u_{i,j}}{h_i^x} - \frac{u_{i,j} - u_{i-1,j}}{h_{i-1}^x}\right) \\ &\quad + \left(\frac{2}{h_j^y + h_{j-1}^y}\right)\left(\frac{u_{i,j+1} - u_{i,j}}{h_j^y} - \frac{u_{i,j} - u_{i,j-1}}{h_{j-1}^y}\right) = f_{ij}. \end{aligned} \tag{2.6}$$

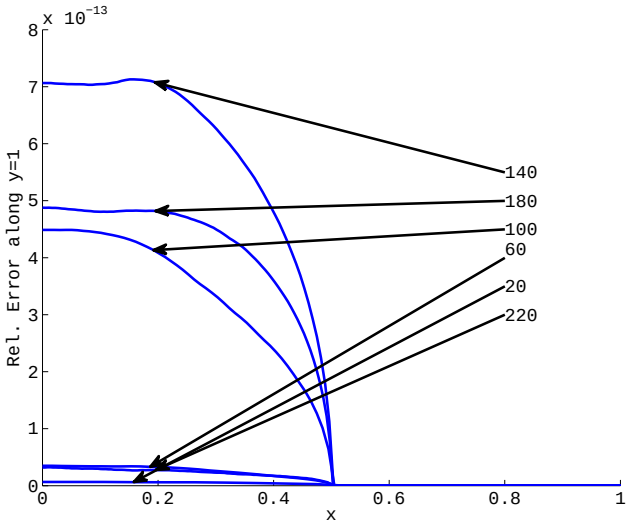

Figure 1: Relative error near the $y = 1$ boundary for varying grid sizes when there is a change in boundary condition type at $x = 0.5$ from Neumann to Dirichlet.

Similarly, for the Poisson equation in polar coordinates (r, ϕ) given by

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \phi^2} = f(r, \phi), \tag{2.7}$$

the approximate solution is given by $u_{i,j} \simeq u(r_j, \phi_i)$, $i = 1, \ldots, L$, $j = 1, \ldots, M$. The resulting discretization is given by

$$\begin{aligned}\left(\frac{2r_j^2}{h_j^r + h_{j-1}^r}\right)\left(\frac{u_{i,j+1} - u_{i,j}}{h_j^r} - \frac{u_{i,j} - u_{i,j-1}}{h_{j-1}^r}\right) + 2r_j\left(\frac{u_{i,j+1} - u_{i,j-1}}{h_k^r + h_{j-1}^r}\right) \\ + \left(\frac{2}{h_i^\phi + h_{i-1}^\phi}\right)\left(\frac{u_{i+1,j} - u_{i,j}}{h_i^\phi} - \frac{u_{i,j} - u_{i-1,j}}{h_{i-1}^\phi}\right) = r_j^2 f_{i,j}.\end{aligned} \tag{2.8}$$

Finally, in spherical coordinates (ρ, θ, ϕ), where θ is azimuthal angle and ϕ is a polar angle, the Poisson equation has the form

$$\frac{1}{\rho^2}\frac{\partial}{\partial \rho}\left(\rho^2\frac{\partial u}{\partial \rho}\right) + \frac{1}{\rho^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta\frac{\partial u}{\partial \theta}\right) + \frac{1}{\rho^2 \sin^2\theta}\frac{\partial^2 u}{\partial \phi^2} = f(\rho, \theta, \phi). \tag{2.9}$$

the approximate solution is given by $u_{i,j,k} \simeq u(\rho_k, \theta_j, \phi_i)$, $i = 1, \ldots, L$, $j = 1, \ldots, M$, $k = 1, \ldots, N$. The discretized version of PDE (2.9) is given by

$$\begin{aligned}&\left(\frac{2\rho_k^2 \sin^2\theta}{h_k^\rho + h_{k-1}^\rho}\right)\left(\frac{u_{i,j,k+1} - u_{i,j,k}}{h_k^\rho} - \frac{u_{i,j,k} - u_{i,j,k-1}}{h_{k-1}^\rho}\right) \\ &+2\rho_k \sin^2\theta\left(\frac{u_{i,j,k+1} - u_{i,j,k-1}}{h_k^\rho + h_{k-1}^\rho}\right) \\ &+\left(\frac{2\sin^2\theta}{h_j^\theta + h_{j-1}^\theta}\right)\left(\frac{u_{i,j+1,k} - u_{i,j,k}}{h_j^\theta} - \frac{u_{i,j,k} - u_{i,j-1,k}}{h_{j-1}^\theta}\right) \\ &+\sin\theta\cos\theta\left(\frac{u_{i,j+1,k} - u_{i,j-1,k}}{h_j^\theta + h_{j-1}^\theta}\right) \\ &+\left(\frac{2}{h_i^\phi + h_{i-1}^\phi}\right)\left(\frac{u_{i+1,j,k} - u_{i,j,k}}{h_i^\phi} - \frac{u_{i,j,k} - u_{i-1,j,k}}{h_{i-1}^\phi}\right) = \rho_k^2 \sin^2\theta f_{i,j,k}.\end{aligned} \tag{2.10}$$

2.3 Singularity Points in Polar and Spherical Domains

In polar and spherical domains, the finite difference formulas for the Laplacian cannot be directly used for internal domain domain points corresponding to zeroes of the Jacobians $J_2 = D(x,y)/D(r,\phi) = r$ and $J_3 = D(x,y,z)/D(\rho,\theta,\phi) = \rho^2 \sin\theta$.

In particular, for the polar and spherical domains, a breakdown occurs at points with $r = 0$ ($\rho = 0$). In the spherical domain, another breakdown occurs for points where $\theta = \{0, \pi\}$. A solution to this problem is found by recalling to the definition of the Laplacian as the divergence of the gradient. Using the fact that the divergence of a vector field $\mathbf{q} \in \mathbb{R}^2$ or $\mathbf{w} \in \mathbb{R}^3$ is given by, respectively,

$$\operatorname{div} \mathbf{q} = \lim_{S \to 0} \frac{1}{S} \int_{\partial S} (\mathbf{q} \cdot \mathbf{n})\, d\ell, \quad \operatorname{div} \mathbf{w} = \lim_{V \to 0} \frac{1}{V} \iint_{\partial V} (\mathbf{w} \cdot \mathbf{n})\, dS, \tag{2.11}$$

one can derive formulas that correctly approximate the Laplacian for points where the Jacobian is singular. In particular, in polar coordinates at $r = 0$, (2.11) yields a finite-difference relation

$$u_{1,j} = \frac{1}{N_\phi} \sum_{j=1}^{N_\phi} u_{3,j} - \left(\frac{(h_r)_i}{2} \right) f(0,\phi) \tag{2.12}$$

determining the value of u in the center of the disk. Note that conditions

$$u_{1,1} = u_{1,2} = \ldots = u_{1,L} \tag{2.13}$$

must be added because the center value is independent of the polar angle ϕ.

For spherical coordinates at $r = 0$, the divergence of the gradient yields,

$$\frac{L(M-1)\Delta A}{h_1^r \Delta V} u_{i,j,1} = \frac{1}{\Delta V} \left(\sum_{i=1}^{L} \sum_{j=1}^{M} \frac{u_{i,j,2} \Delta A}{h_1^r} + \frac{(u_{i,1,2} - u_{i,1,1} + u_{i,M,2} - u_{i,M,1}) \Delta A'}{h_1^r} + \right) - f(0,\theta,\phi) \tag{2.14}$$

where

$$\Delta A = \left(\frac{h_1^r}{2} \right)^2 h_j^\theta h_i^\phi \sin\theta_j, \quad \Delta A' = 2\pi h_1^r (1 - \cos h_i^\phi), \quad \Delta V = \frac{4}{3}\pi \left(\frac{1}{2} h_1^r \right)^3,$$

and the value of u at the origin is θ- and ϕ-independent:

$$u_{i,j,1} = u_{1,1,1}, \quad i = 1, \ldots, L; \quad j = 1, \ldots, M. \tag{2.15}$$

Finally, for spherical coordinates at $\theta = 0$, the divergence of the gradient yields,

$$u_{i,1,k} \left(\frac{S_{up} + S_{down}}{h_k^r \widetilde{\Delta V}} \right) = \left(\frac{u_{i,1,k+1} S_{up} + u_{i,1,k-1} S_{down}}{h_k^r \widetilde{\Delta V}} \right) + \frac{\widetilde{\Delta A}}{\widetilde{\Delta V}} \sum_{i=1}^{N_\phi} \frac{u_{i,3,k} - u_{i,1,k}}{2 r_k h_1^\theta} - f(r,\theta,\phi) \tag{2.16}$$

where

$$S_{up} = \frac{1}{2}\pi h_1^\theta \left(r_k + \frac{h_k^r}{2} \right)^2, \quad S_{down} = \frac{1}{2}\pi h_1^\theta \left(r_k - \frac{h_k^r}{2} \right)^2, \quad \widetilde{\Delta A} = \frac{1}{2} r_k h_k^r h_i^\phi \sin\theta_j,$$

$$\widetilde{\Delta V} = \frac{\pi}{3} \left(\frac{1}{2} h_1^\theta \right)^2 \left[\left(r_k + \frac{h_k^r}{2} \right)^3 - \left(r_k - \frac{h_k^r}{2} \right)^3 \right].$$

[For $\theta = \pi$, the expressions are the same, with $u_{i,1,q} \to u_{i,M,q}$, $u_{i,3,q} \to u_{i,M-2,q}$, and $h_1^\theta \to h_M^\theta$.]

2.4 Matrix Structure

We now illustrate the matrix structure arising from an application of finite-difference discretization of the mixed Poisson BVP (1.2) in Cartesian and polar coordinates in two dimensions, and spherical coordinates in three dimensions.

In two dimensions, for all points of a rectangular domain or a disk except for the center of the disk, the discretization of the Poisson equation as well as Dirichlet and Neumann boundary conditions at any point indexed by (i, j) can be written in a general form

$$au_{i,j} + bu_{i+1,j} + cu_{i-1,j} + du_{i,j+1} + eu_{i,j-1} = F_{i,j} \tag{2.17}$$

with the appropriate choice for coefficients a, b, c, d, and e. Here $F_{i,j} = f_{i,j}$ for internal mesh points; it also includes terms associated with functions $v_D(x)$ or $v_N(x)$ at boundary mesh points.

In Cartesian coordinates, the discretization yields a system of linear algebraic equations (LAE) containing $\mathcal{N} = LM$ equations. In order to write equations (2.17) in the matrix form

$$A\mathbf{u} = \mathbf{b}, \tag{2.18}$$

one needs to re-index the matrix of unknown approximate values $u_{i,j}$ into a vector of length $\mathcal{N}$; this is done according to the formula

$$\mathbf{u} = [u_{1,1}, \ldots, u_{L,1}, \quad u_{1,2}, \ldots, u_{L,2}, \quad \ldots, \quad u_{1,M}, \ldots, u_{L,M}]^T. \tag{2.19}$$

Similar operations are performed for polar and spherical coordinates in order to bring the linear system to the form (2.18).

Each one-dimensional discretization yields a tridiagonal matrix A_1, the discretization in two dimensions yields a block-diagonal matrix $A = A_2$ which consists of matrices A_1 on the main diagonal, with additional layer-interaction elements (two extra nonzero diagonals), illustrated clearly in Figure 2 (left). [The three matrix structures shown in Figure 2 were observed using the `Matlab spy` function.]

For a two-dimensional disk domain in polar coordinates, the matrix structure is rather similar, with the addition of singularity conditions (2.12), (2.13) which produce extra relations seen as a horizontal and a vertical bar in the matrix plot in Figure 2 (middle).

Similarly, in three dimensions (spherical coordinates), the corresponding matrix $A = A_3$ is a block-diagonal matrix with A_2-type blocks on the main diagonal and two additional nonzero layer-interaction diagonals. The matrix structure is shown in Figure 2 (right); it also involves horizontal and vertical bars corresponding to relations (2.14), (2.15), and (2.16) at singular mesh points.

Since the number of nonzero matrix entries depends on the number of unknowns linearly, and the matrix size – quadratically, the matrix structures become increasingly more sparse as the number of mesh points increases. It is therefore paramount that one utilizes a matrix solver that is efficient in solving (2.18) with large sparse matrices.

Another important characteristic of the A matrix that needs to be discussed is the condition number

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|,$$

where $\|A\|$ is some matrix norm. The condition number is commonly viewed as an upper bound of the error amplification coefficient for the problem (2.18), according to the formula

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right),$$

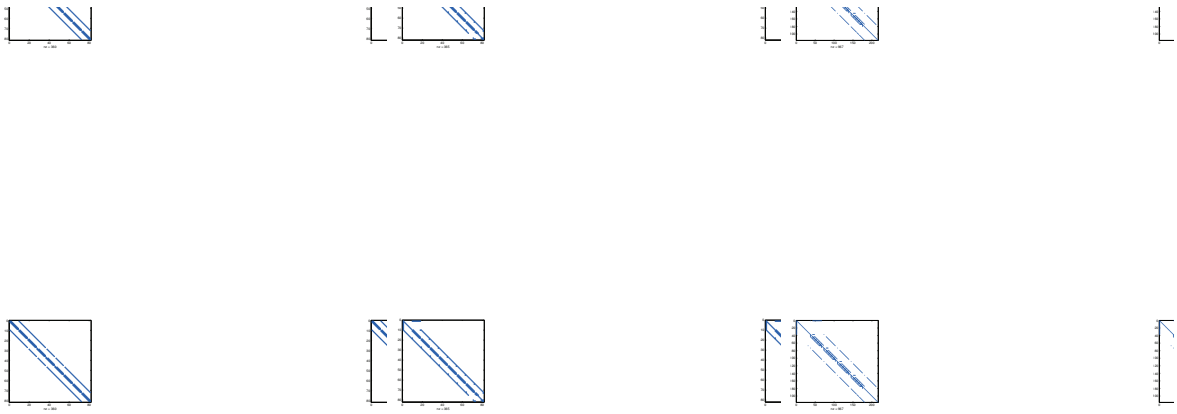

Figure 2: Left: matrix structure for a 9×9 Cartesian domain. Middle: matrix structure for a 9×9 polar domain. Right: matrix structure for a $6 \times 6 \times 6$ spherical domain. [In each caption, nz stands for the number of nonzero matrix elements.]

where δA and $\delta \mathbf{b}$ are errors (resulting from roundoff and/or truncation errors) in the matrix and the right-hand side, and $\delta \mathbf{x}$ is the resulting error in the solution.

For matrices of the structure described above, condition numbers computed using the 1-norm can reach the order of $10^5 - 10^{10}$, growing with the number of mesh points. An example of a computation of relative errors and condition numbers for a Cauchy problem in a unit square with exact solution $u(x, y) = \exp[-300((x - 0.45)^2 + (y - 0.61)^2)$ is given in Fig. 3, where calculations have been performed for increasing numbers of points N in each direction, with no mesh refinement. The condition numbers were computed using `Matlab condest` 1-norm condition number estimate.

The high matrix condition numbers do not provide limitations for the numbers of mesh points in the computations, since a significantly more severe limitation is provided by memory constraints. Matrix condition numbers cannot be improved by preconditioners for the main `mldivide` routine since it is based Gaussian elimination; for the iterative solver, the LU-decomposition is used as a preconditioner.

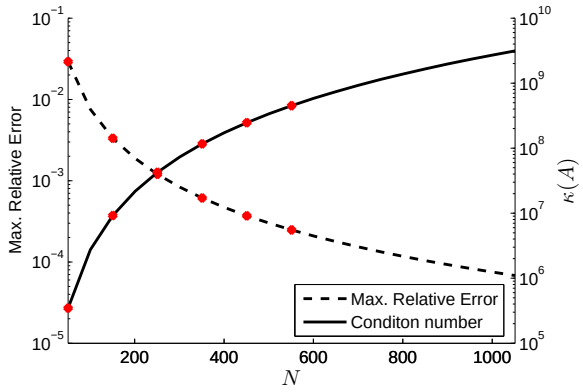

Figure 3: Condition number and maximum relative error values versus the number N of mesh points in each direction, for the Cauchy problem for a Possion equation in a unit square, with a known analytical solution. Black curves correspond to data obtained using `mldivide`; red dots indicate cases when an iterative solver was used. Computations were performed for increasing values of N until memory limits were reached for the `Matlab condest` routine.

2.5 Matlab and Matrix Optimized Code

With `Matlab`'s optimized routines for handling and solving linear matrix equations (2.18), including very large sparse matrices, it was an obvious choice to minimize the amount of programming effort required to develop this solver. By building a finite difference algorithm that avoids `for`-loops and takes advantage of processor-specific fast vector operations implemented in `Matlab`, computation time was dramatically decreased.

With multi-core processor computers becoming the norm, `Matlab` has developed its vector-handling routines to be parallelized across multiple cores, thus further decreasing the computational time required to obtain a solution. In particular, run examples discussed in the current paper were tested on a quad-core `Intel` processor, and in the course of the solution, the computational load was approximately equally distributed between the four cores.

2.6 Mesh Refinement

For some problems of the type (1.2), it may be necessary to employ mesh refinement. In particular, MFPT problems (1.1) involved small boundary traps (Dirichlet domains) that needed to be adequately resolved, with at least 100 mesh points per trap. At the same time, with common desktop hardware limitations of standard laboratory computers, it was only possible to employ $N \sim 10^6$ data points, depending on the domain geometry. For example, for a unit square, computations were performed with $L = M = 1300$; for a unit sphere, values around $L = 250, M = N = 100$ were used.

Within the routines presented in the current paper, the refinement is done by applying a non-linear scaling to the numerical mesh in any or every chosen direction, such that mesh points are concentrated around expected regions of interest.

An advantage gained by using the above mesh refinement procedure in terms of computing time and accuracy can be illustrated by the following example. Consider an exact sharp-spike type solution

$$u(x, y) = e^{-10^6[(x-0.5)^2+(y-0.5)^2]} \tag{2.20}$$

and its corresponding Dirichlet problem for the Poisson equation in the unit square. Since the spike is located at the square center, a cubic refinement curve

$$\tilde{x} = 4(x - 0.5)^3 + 0.5$$

is used in both spatial directions. Fig. 4 shows the maximum relative error for the numerical solution with mesh refinement with a mesh size of 100×100 and the maximum error for the solution without mesh refinement on an $N \times N$ mesh, as well as the the amount of time required to compute each solution. The error level of the solution using a mesh with $N = 100$ and mesh refinement is reached by a solution without mesh refinement only for $N \sim 520$; the latter also requires significantly more memory and time.

The requirement of mesh refinement is especially critical in the three-dimensional domain of the unit sphere. For example, for the MFPT problem with boundary conditions involving three traps (Dirichlet regions) centered at polar angles $\phi = \pi/3$, π, $5\pi/3$ on the equator, the refinement curves are shown in Fig. 5. Places of almost-zero slopes correspond to multiple points of a homogeneous mesh being concentrated near regions of interest. For the above spherical example, with $L = 250$ points in the polar ϕ-direction and $M = N = 100$ in ρ- and θ-directions, small Dirichlet (trap) boundary regions we resolved at 9 points per trap when no mesh refinement was used, and at 799 points per trap when mesh refinement shown in Fig. 5 was applied.

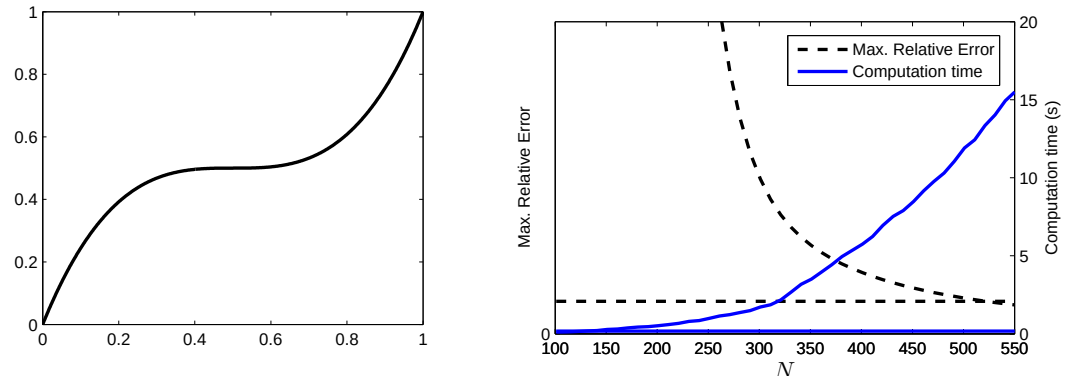

Figure 4: Left: the mesh refinement curve for x- and y-directions. Right: comparison of maximum relative error (dashed curves) and computational time required (solid curves) for solution with a 100×100 mesh with mesh refinement (horizontal lines) and solutions using an $N \times N$ mesh without mesh refinement, $100 \le N \le 550$.

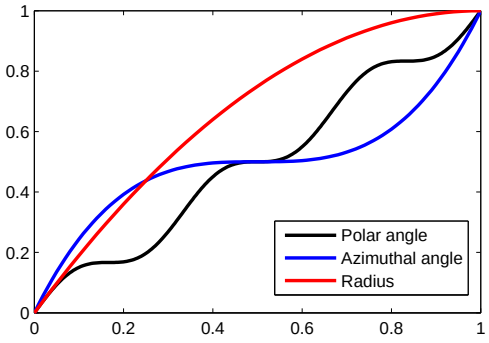

Figure 5: Mesh refinement for a sphere with regions of interest centered at polar angles $\phi = \pi/3$, π, $5\pi/3$ on the equator. The horizontal axis corresponds to input (homogeneous) mesh and the vertical axis corresponds to output (refined) mesh.

Several run examples using mesh refinement in two and three dimensions are presented in the sections below.

It is important to note that the mesh refinement algorithm used in the current work is not a local refinement; in fact, the refinement propagates through the domain, which yields extra points, or points that appear to have no function (see, e.g., left plot in Fig. 7, and upper left plot in Fig. 10). Variants of finite-difference methods that used local mesh refinement were considered by the authors. It was however found that even though the number of points in local mesh refinement algorithms is smaller, computations in fact take significantly longer, due to the fact that a local finite-difference refinement patch destroys the block matrix structure discussed in Section 2.4. Hence the current (somewhat nonlocal) mesh refinement algorithm was adopted as a more efficient one from the practical point of view.

3 Rectangular Domain in Two Dimension

Consider a mixed Poisson problem (1.2) for the rectangular domain $\Omega = \{(x, y) \mid 0 \le x \le L_x,\ 0 \le y \le L_y\}$. Denote the numbers of mesh points in x- and y-directions by L and M, respectively.

3.1 Program Sequence

In order to numerically solve the problem (1.2) for a rectangle in cartesian coordinates, one needs to perform the following steps.

1. Create a new folder and copy all files of some example that solves the Poisson equation in a rectangle into that new folder.

2. Modify the file `forcing.m`: change the line `f_mat(i,j)=Q` where $Q(x_i, y_j)$ is the desired forcing function.

3. Modify the file `dirichlet_boundary.m`: change the `if`-conditions as needed, and change the lines `diriBC(i,j)=Q` so that $Q = v_D(x_i, y_j)$ specifies the desired Dirichlet boundary condition of the problem (1.2).

4. Modify the file `neumann_boundary.m`: change the `if`-conditions as needed, and change the lines `neuBC(i,j)=Q` so that $Q = v_N(x_i, y_j)$ specifies the desired Neumann boundary condition of the problem (1.2).

5. Modify the files `x_refine_function.m` and `y_refine_function.m` to introduce necessary mesh refinement. The input parameter `in` is a vector that will contain homogeneous mesh points; the output parameter `out` has to either equal `in` (no mesh refinement) or a function that acts on `in` to concentrate/rarify mesh where needed. Each refinement function must be a monotone increasing function. It is the user's responsibility to satisfy the conditions

   ```
   in(1)=out(1),   in(end)=out(end)
   ```
   (3.1)

   in order to preserve the square side lengths.

   [Often it is useful to plot `out` as a function of `in` using Matlab `plot(in, out)`, to obtain a curve similar to those in Fig. 5.]

6. Run the solver routine:

   ```
   [xs ys v relres iter resvec] = rectangle_2d_poisson(
       x_max,y_max,num_xs,num_ys,'x_refine_function','y_refine_function',
       'dirichlet_boundary','neumann_boundary','forcing',useiter)
   ```

   where `x_max` and `y_max` are the domain sizes L_x, L_y in the `x`-and `y`-directions respectively; `num_xs` and `num_ys` are the numbers of mesh points L, M in the corresponding directions; `x_refine_function.m` and `y_refine_function.m` are the filenames of external `Matlab` functions for mesh refinement; `dirichlet_boundary.m` and `neumann_boundary.m` are the filenames of external `Matlab` functions that generate the Dirichlet and Neumann boundary conditions; `forcing.m` is the name of an external `Matlab` function that generates the forcing term at every point of the domain; `useiter` is an optional parameter (default value 0) which, if set to a nonzero value, forces the solver to use an iterative method to solve the sparse linear system.

   Details about the parameters passed to the `rectangle_2d_poisson` solver, in particular, formats of the external functions, are discussed below within the run examples (Section 3.2).

   The normal output of the solver consists of a matrix of approximate solutions `v` of the problem (1.2) at mesh points stored in matrices `xs` and `ys`. The solution is ready for plotting using the routine

   ```
   surf(xs,ys,v);
   ```

When the parameter `useiter` is set to a nonzero value, an iterative solver for a linear sparse system is used, and the output values `relres`, `iter` and `resvec` are the final residual, the number of iterations used, and a vector of the history of the residuals, respectively.

Remark 3.1. The solver applies the following rules to determine which boundary condition to use at each boundary point.

- If at a given boundary point, a Dirichlet boundary condition is specified, then it is used, and the Neumann boundary condition is ignored (if specified at all).
- If at a given boundary point, a Neumann boundary condition needs to be specified, the value provided for `diriBC` for that point in the file `dirichlet_boundary.m` must be set to `NaN`.

The above rules let the user avoid duplicating `if`-conditions when the problem involves a mixture of Dirichlet and Neumann boundary parts.

3.2 Run Examples

3.2.1 Example R1

As a first run example for the mixed Poisson problem (1.2) for a rectangular domain, we let $L_x = 1, L_y = 2$, with $L = M = 100$ mesh points in each direction, and no mesh refinement.

We define the solution $v(x, y)$ to be a known "chair-shaped" function given by

$$v(x,y) = \left[3\exp\{-100(x-0.95)^2 - 10(y-1)^2\} + 1\right]\cos\left[2\pi(y-1)(x-0.5)\right]. \tag{3.2}$$

Suppose the function (3.2) solves the problem (1.2) with Neumann boundary conditions at $x = 0$ and $y = 0$, and Dirichlet boundary conditions at $x = 1$ and $y = 1$. The boundary conditions v_N and v_D and forcing $f(x, y)$ are computed by substituting the expression (3.2) into the problem (1.2).

The `Matlab` files for the boundary conditions, forcing, and mesh refinement functions are given in Appendix A.

The solver is run by the command

```
[xs ys v relres iter resvec] = rectangle_2d_poisson(
    1, 2, 100, 100, 'x_refine_function','y_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing')
```

and the solution is plotted using the command `surf(xs,ys,v,'EdgeColor','none');`

The obtained numerical solution v_{ij} and the error (absolute difference between the exact solution (3.2) and the numerical solution) are shown in Fig. 6.

[Note that though the solution $v(x, y)$ (3.2) is symmetric with respect to the plane $y = 1$, the numerical solution and error plot is not, due to the asymmetry of the boundary conditions.]

3.2.2 Example R2

The second run example corresponds to the narrow escape problem (1.1) for a square domain with $L_x = L_y = 1$, $D = 1$, and almost completely Neumann boundary, except for the two Dirichlet traps of length 0.02 on the side $x = 0$, centered at $y = 0.25$ and $y = 0.75$.

For the numerical solution, $L = M = 100$ mesh points in each direction were taken, with mesh refinement in order to resolve the small traps. The mesh refinement functions are given in Appendix B; the mesh and the numerical solution are shown in Fig. 7.

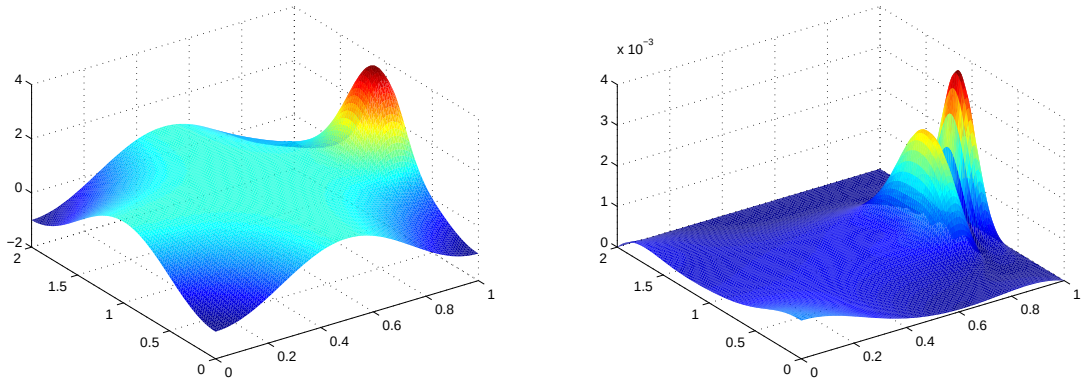

Figure 6: Left: numerical solution for the mixed Poisson problem in the rectangular domain (Section 3.2.1). Right: error plot [absolute difference between the exact and the numerical solution].

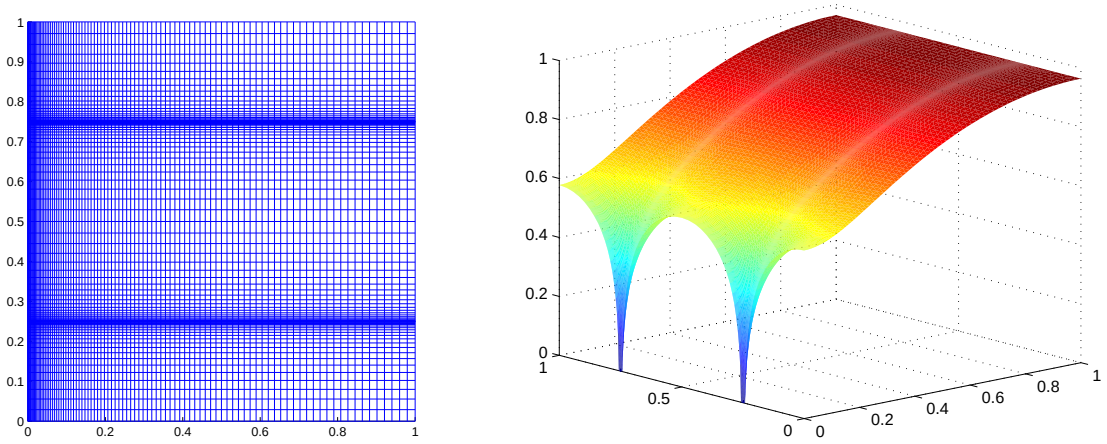

Figure 7: Left: the refined mesh. Right: the numerical solution of the narrow escape problem for the square with two traps (Section 3.2.2).

4 Disk Domain in Two Dimensions

In order to numerically solve the problem (1.2) for a disk of radius R in polar coordinates, the following steps need to be taken.

4.1 Program Sequence

1. Create a new folder and copy all files of some example that solves the Poisson equation in a disk into that new folder.

2. Modify the file `forcing.m`: change the line `f_mat(i,j)=Q` where $Q(r_j, \phi_i)$ is the desired forcing function.

3. Modify the file `dirichlet_boundary.m`: change the `if`-conditions as needed, and change the lines `diriBC(i)=Q` so that $Q = v_D(\phi_i)$ specifies the desired Dirichlet boundary condition of the problem (1.2). [See also Remark 3.1.]

4. Modify the file `neumann_boundary.m`: change the `if`-conditions as needed, and change the

lines `neuBC(i)=Q` so that $Q = v_N(\phi_i)$ specifies the desired Neumann boundary condition of the problem (1.2). [See also Remark 3.1.]

5. Modify the files `r_refine_function.m` and `phi_refine_function.m`, similarly to the case of the rectangle. Each refinement function must be a monotone increasing function. It is the user's responsibility to satisfy the conditions (3.1) to preserve the domain sizes for both variables. Note that for the function `phi_refine_function.m`, one must have

   `in(1)=out(1)=0,  in(end)=out(end)=`2π.

6. Run the solver routine:

```
[xs ys v rs phis relres iter resvec] = polar_2d_poisson(
    R_max,num_rs,num_phis,'r_refine_function','phi_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing',useiter)
```

   where `R_max` is the disk radius R, `num_rs` and `num_phis` are the numbers of mesh points L, M in the radial and angular polar directions; `r_refine_function.m` and `phi_refine_function.m` are the filenames of external `Matlab` functions for mesh refinement. For the output values `v` of the numerical solution, the routine outputs both the matrices of corresponding polar coordinates `rs`, `phis` and the corresponding Cartesian coordinates `xs`, `ys`. The rest of input and output parameters is identical to those listed in Section 3.1.

4.2 Run Examples

4.2.1 Example D1

As a first run example, we take a known exact solution

$$v(r, \phi) = r^3 \sin^2 \phi. \tag{4.1}$$

of a purely Neumann problem in the unit disk ($R = 1$):

$$\begin{aligned} &\Delta v(r, \phi) = -r(5\cos^2\phi - 7), \quad 0 \le r < 1, \quad 0 \le \phi < 2\pi, \\ &\partial_r v(1, \phi) = 3\sin^2\phi, \qquad\qquad 0 \le \phi < 2\pi. \end{aligned} \tag{4.2}$$

The solution of the purely Neumann problem (4.2) is not unique, since an arbitrary constant can be added to it. For such problems, the numerical solver finds a numerical solution whose minimum value is zero.

For the numerical solution, we take a rather coarse mesh with $L = 50$ mesh points in the radial direction and $M = 100$ points in the polar direction, with no mesh refinement in the angle, and radial mesh refinement according to the law $r' = r(2 - r)$ that concentrates points at the boundary $r = R = 1$. The resulting numerical mesh, numerical solution, and the absolute error are shown in Fig. 8.

4.2.2 Example D2

As a second example, consider a narrow escape problem (1.1) in a unit disk ($0 \le r \le R = 1$) in polar coordinates, with mean first passage time for a particle starting at r, ϕ given by $v(r, \phi)$. Let $D = 1$, and let the absorbing boundary portion $\partial\Omega_a$ consist of a union of seven arcs of lengths $\ell = 0.1$ centered at

$$\phi = 0, \quad \frac{3\pi}{7}, \quad \frac{25\pi}{42}, \quad \frac{3\pi}{4}, \quad \pi, \quad \frac{3\pi}{2}, \text{ and } \frac{13\pi}{8}.$$

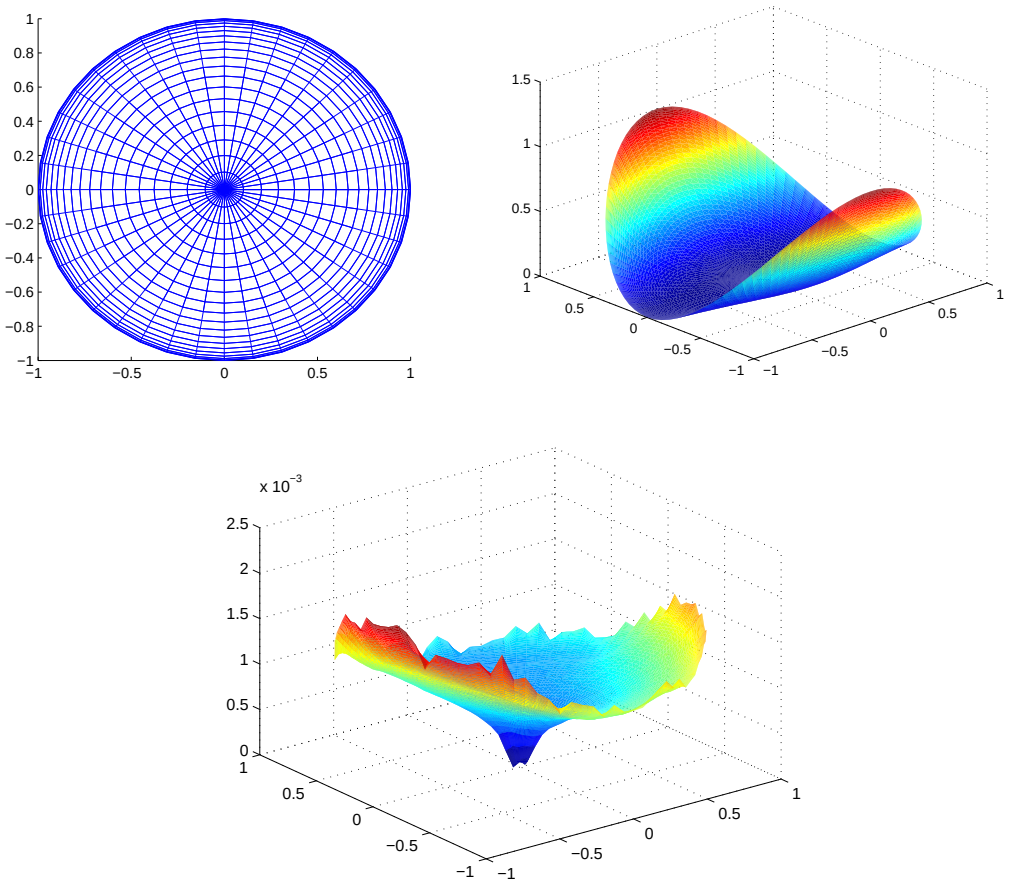

Figure 8: Left to right: numerical mesh, numerical solution, and absolute error for the Neumann Poisson problem in a disk (Section 4.2.1).

On the remaining (reflective) part of the boundary, $\partial\Omega_r = \{(r = 1, \phi)\} \setminus \partial\Omega_a$, the homogeneous Neumann boundary condition is prescribed: $\partial_r v(1, \phi) = 0$.

The current example uses no mesh refinement in the polar angle, and radial mesh refinement according to the law $r' = r(2-r)$. The `Matlab` functions corresponding to the boundary conditions, mesh refinement, and forcing, are given in Appendix C.

A numerical solution generated by the command

```
[xs ys v rs phis relres iter resvec] = polar_2d_poisson(
    1,200,400,'r_refine_function','phi_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing')
```

using $L = 200$ points in the radial direction and $M = 400$ points in the polar direction is shown in Fig. 9.

5 Spherical Domain in Three Dimensions

Finally, in order to numerically solve the problem (1.2) for a sphere of radius R in spherical coordinates (ρ, θ, ϕ), one needs to perform the following steps.

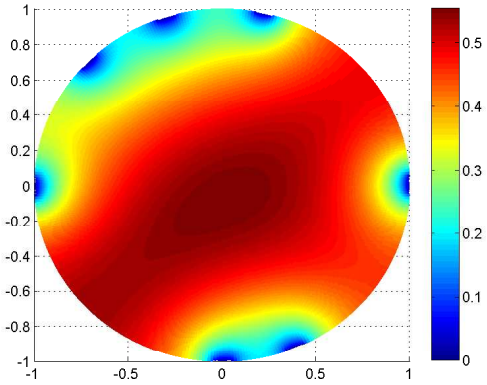

Figure 9: Numerical solution of a narrow escape problem in a disk, with seven absorbing traps of equal lengths $\ell = 0.1$ (Section 4.2.2).

5.1 Program Sequence

1. Create a new folder and copy all files of some example that solves the Poisson equation in a sphere into that new folder.

2. Modify the file `forcing.m`: change the line `f_mat(i,j,k)=Q` where $Q(\rho_k, \theta_j, \phi_i)$ is the desired forcing function.

3. Modify the file `dirichlet_boundary.m`: change the `if`-conditions as needed, and change the lines `diriBC(i,j)=Q` so that $Q = v_D(\theta_j, \phi_i)$ specifies the desired Dirichlet boundary condition of the problem (1.2). [See also Remark 3.1.]

4. Modify the file `neumann_boundary.m`: change the `if`-conditions as needed, and change the lines `neuBC(i,j)=Q` so that $Q = v_N(\theta_j, \phi_i)$ specifies the desired Neumann boundary condition of the problem (1.2). [See also Remark 3.1.]

5. Modify the files `r_refine_function.m`, `phi_refine_function.m`, and `theta_refine_function.m`, similarly to the case of the rectangle. It remains the user's responsibility to satisfy the conditions (3.1) for each refinement function to preserve the domain sizes for all three variables. Note that for the function `theta_refine_function.m`, one must have

   `in(1)=out(1)=0`, `in(end)=out(end)=`π,

   and for the function `phi_refine_function.m`, one must have

   `in(1)=out(1)=0`, `in(end)=out(end)=`2π.

6. Run the spherical solver:

```
[rs thetas phis v xs ys zs relres iter resvec] = sphere_3d_poisson(
    R_max,num_rs,num_thetas,num_phis,
    'r_refine_function','theta_refine_function.m', 'phi_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing',useiter)
```

   where `R_max` is the sphere radius R. The numerical solution `v` is output in the form of a three-dimensional array, together with the corresponding values of spherical coordinates `rs`, `thetas`, `phis`, and Cartesian coordinates `xs`, `ys`, `zs`, and is ready for plotting. The rest of the input and output parameters is analogous to parameters described in Sections 3.1 and 5.1.

   Unlike the two-dimensional solvers, for the spherical solver, the default value of `useiter=1` is used, i.e., an iterative sparse matrix solver is employed by default.

5.2 Run Example

As a run example, we let $\varepsilon = 0.1$, and consider the following mixed Dirichlet-Neumann Poisson problem for $v(x)$ in a unit sphere in spherical coordinates $x = (\rho, \theta, \phi)$.

$$\begin{aligned}
&\Delta v = 0, && \rho < 1, \\
&v = 1, && \rho = 1, \quad \theta \in \left[\tfrac{\pi}{2} - \varepsilon, \tfrac{\pi}{2} + \varepsilon\right], \quad \phi \in \left[\tfrac{\pi}{3} - \varepsilon, \tfrac{\pi}{3} + \varepsilon\right], \\
&v = -10, && \rho = 1, \quad \theta \in \left[\tfrac{\pi}{2} - \varepsilon, \tfrac{\pi}{2} + \varepsilon\right], \quad \phi \in [\pi - \varepsilon, \pi + \varepsilon], \\
&\partial_\rho v = 0, && \rho = 1, \quad \text{all other } (\theta, \phi).
\end{aligned} \tag{5.1}$$

In this example, the problem (5.1) is solved numerically with $L = 50$ mesh points in the radial direction, $M = 50$ points in the azimuthal direction, and $N = 100$ points in the polar direction, using the command

```
[rs thetas phis u xs ys zs] = sphere_3d_poisson(
    1,50,50,100,
    'r_refine_function','theta_refine_function.m', 'phi_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing')
```

involving simple mesh refinement functions. Boundary condition and mesh refinement function code is listed in Appendix D.

Solution plots and the numerical mesh for the current example are shown in Fig. 8.

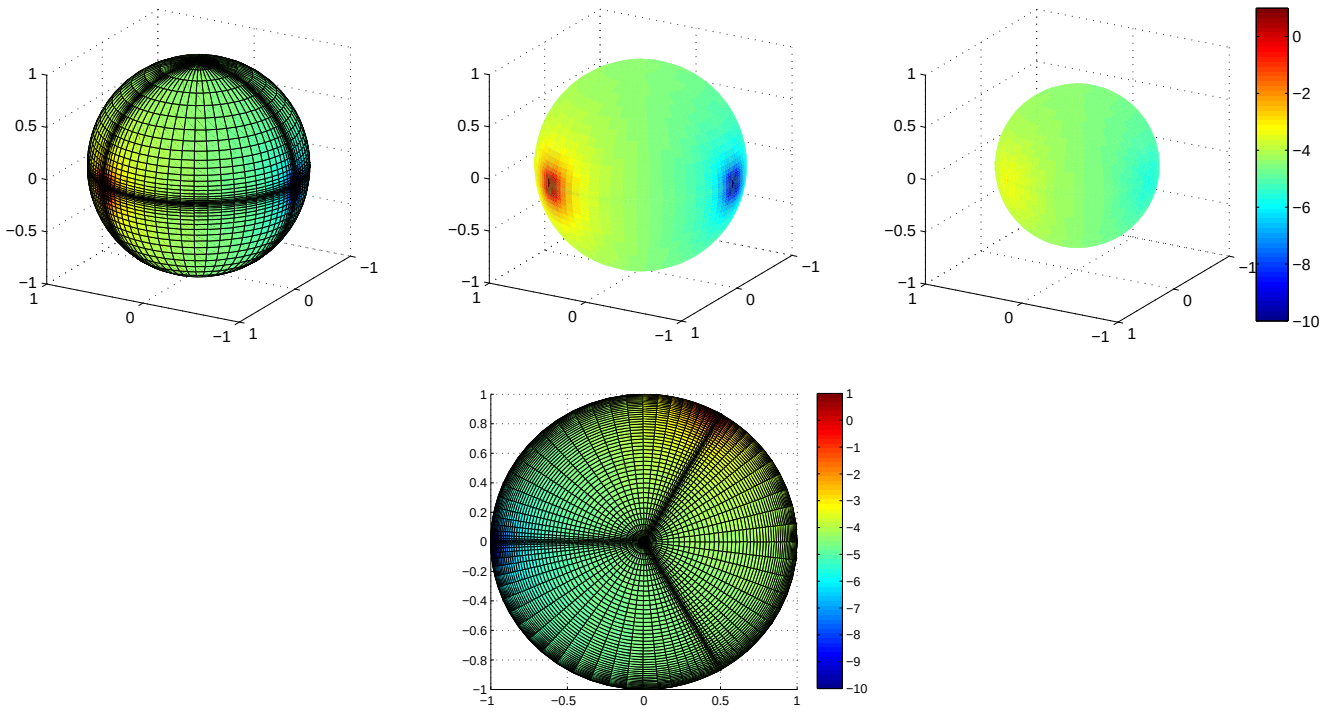

Figure 10: Top: plot of the numerical solution $v(\rho_i, \theta, \phi)$ of the problem (5.1) at radii $\rho_1 = 1, \rho_2 = 0.8, \rho_3 = 0.5$ [with the numerical mesh shown in the left plot). Bottom: the numerical solution $v(\rho, \pi/2, \phi)$ (equatorial cross-section) and the equatorial cross-section of the numerical mesh. (Section 5.2).

6 Conclusions

The presented `Matlab`-based set of functions provides an effective numerical solution of linear Poisson boundary value problems (1.2) involving an arbitrary combination of homogeneous and/or non-homogeneous Dirichlet and Neumann boundary conditions, for a rectangle and a disk in two dimensions, and a spherical domain in three dimensions. Computations exhibit high speeds, numerical stability with respect to mesh size and mesh refinement, and low values of error tolerance even on desktop computers.

The finite-difference numerical method used in the algorithm allows for mesh refinement, position-dependent forcing, and also for effective switching between Dirichlet and Neumann boundary conditions at various portions of the boundary. The numerical method is second-order accurate for homogeneous meshes and first-order accurate for variable-step meshes. Mesh refinement procedures are fully user-controlled.

Within the routines discussed in the current paper, a large sparse linear system for the unknown approximate solution values at mesh points is set up, using `Matlab` specialized processor-effective sparse vector and matrix operations. The use of such vector operations as opposed to nested `for`-loops lead to an increase in performance by a factor sometimes as large as several thousands.

The large sparse system of linear algebraic equations is subsequently solved using the effective `Matlab` backslash (`mldivide`) function, or an iterative `bicg` routine, in cases when `mldivide` runs out of memory.

The routines presented are also compatible with `Octave`, an open-source `Matlab`-like program, except when the iterative solving options (`bicg`, etc.) are used. An `Octave`/`Matlab` comparison for the example of Section 3.2.1 in terms of running time is given in Fig. 11.

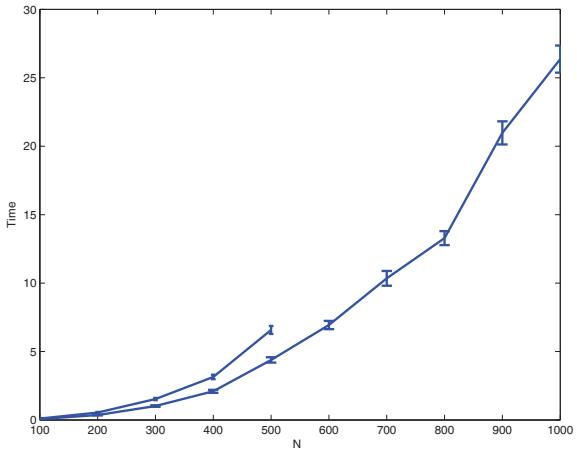

Figure 11: Computational time (seconds) required for `Octave` and `Matlab` to solve the Dirichlet-Neumann problem described in Section 3.2.1, as a function of the number N of mesh points in each direction ($N \times N$ mesh). The shorter curve corresponds to `Octave`.

The `Matlab`-based solvers offer a simple, transparent, and easily adaptable method to solving Poisson problems. The presented routines can be generalized to other systems of curvilinear coordinates using similar considerations and a corresponding set of Lamé factors. It is also straightforward to modify the routines to use Robin boundary conditions if needed.

On multi-processor and/or multi-core systems, programs run in parallel using `Matlab` internal parallelization mechanisms.

Acknowledgements

The authors are grateful to NSERC for research support through a Discovery grant (A.C.) and a USRA fellowship (A.R.).

References

[1] O. Bénichou and R. Voituriez. Narrow escape time problem:time needed for a particle to exit a confining domain through a small window. *Phys. Rev. Lett*, 100:168105, 2008.

[2] A. F. Cheviakov, M. J. Ward, and R. Straube. An asymptotic analysis of the mean first passage time for narrow escape problems: Part ii: The sphere. *Multiscale Modeling & Simulation*, 8(3):836–870, 2010.

[3] D. Holcman and Z. Schuss. Escape through a small opening: Receptor trafficking in a synaptic membrane. *J. Stat. Phys.*, 117(5-6):975–1014, 2004.

[4] W. Liniger and F. Odeh. On the numerical treatment of singularities in solutions of laplace's equation. *Numerical Functional Analysis and Optimization*, 16(3-4):379–393, 1995.

[5] V. S. Lukoshkov. Some electrostatic properties of grid electrodes. *Izv. Akad. Nauk SSSR Ser. Fiz. (in Russian)*, 8:243–247, 1944.

[6] B. Ya. Moizhes. Averaged electrostatic boundary conditions for metallic meshes. *Zh. Tekh. Fiz. (in Russian)*, 25:167–176, 1955.

[7] P. Moon and D. E. Spencer. *Field Theory for Engineers*. D. Van Nostrand Company, 1961.

[8] P. Moon and D. E. Spencer. *Field Theory Handbook*. Springer-Verlag, 1971.

[9] S. Pillay, M. J. Ward, A. Peirce, and T. Kolokolnikov. An asymptotic analysis of the mean first passage time for narrow escape problems: Part i: Two-dimensional domains. *Multiscale Modeling & Simulation*, 8(3):803–835, 2010.

[10] Z. Schuss, A. Singer, and D. Holcman. The narrow escape problem for diffusion in cellular microdomains. *PNAS*, 104(41):16098–16103, 2007.

A Function Code for the Run Example of Section 3.2.1

```
function [FRONT, BACK, LEFT, RIGHT]=dirichlet_boundary(xs,ys)
        N=length(xs);
        M=length(ys);
        LEFT=zeros(1,M);
        RIGHT=zeros(1,M);
        FRONT=zeros(1,N);
        BACK=zeros(1,N);

        %FRONT DIRICHLET BOUNDARY (y=0)
        for i=1:1:N
            x=xs(i);
            FRONT(i)=NaN;
        end

        %BACK DIRICHLET BOUNDARY (y=Ly)
        for i=1:1:N
            x=xs(i);
```

```
            BACK(i)=cos(2*(x-.5)*pi)*(3*exp(-100*(x-.95)^2-10)+1);
        end

        %LEFT DIRICHLET BOUNDARY (x=0)
        for j=1:1:M
            y=ys(j);
            LEFT(j)=NaN;
        end

        %RIGHT DIRICHLET BOUNDARY (x=Lx)
        for j=1:1:M
            y=ys(j);
            RIGHT(j)=cos(1.0*(y-1)*pi)*(3*exp(-.2500-10*(y-1)^2)+1);
        end
end
```

```
function [FRONT, BACK, LEFT, RIGHT]=neumann_boundary(xs,ys)
        N=length(xs);
        M=length(ys);
        LEFT=zeros(1,M);
        RIGHT=zeros(1,M);
        FRONT=zeros(1,N);
        BACK=zeros(1,N);

        %FRONT NEUMANN BOUNDARY (y=0)
        for i=1:1:N
            x=xs(i);
            FRONT(i)=-2*sin(-2*(x-.5)*pi)*(x-.5)*pi*(3*exp(-100*(x-.95)^2-10)+1)+60*
                cos(-2*(x-.5)*pi)*exp(-100*(x-.95)^2-10);
        end

        %BACK NEUMANN BOUNDARY (y=Ly)
        for i=1:1:N
            x=xs(i);
            BACK(i)=0;
        end

        %LEFT NEUMANN BOUNDARY (x=0)
        for j=1:1:M
            y=ys(j);
            LEFT(j)=-2*sin(-1.0*(y-1)*pi)*(y-1)*pi*(3*exp(-90.2500-10*(y-1)^2)+1)
                +570.00*cos(-1.0*(y-1)*pi)*exp(-90.2500-10*(y-1)^2);
        end

        %RIGHT NEUMANN BOUNDARY (x=Lx)
        for j=1:1:M
            y=ys(j);
            RIGHT(j)=0;
        end
end
```

```
function [fmat]=forcing(xs,ys)
    N=length(xs);
    M=length(ys);
    fmat=zeros(N,M);
    for j=1:1:M
        y=ys(j);
        for i=1:1:N
            x=xs(i);
            fmat(i,j)=-4*cos(2*(y-1)*(x-.5)*pi)*(y-1)^2*pi^2*(3*exp(-100*(x-.95)
                ^2-10*(y-1)^2)+1)-12*sin(2*(y-1)*(x-.5)*pi)*(y-1)*pi*(-200*x+190.00)*
                exp(-100*(x-.95)^2-10*(y-1)^2)-660*cos(2*(y-1)*(x-.5)*pi)*exp(-100*(x
                -.95)^2-10*(y-1)^2)+3*cos(2*(y-1)*(x-.5)*pi)*(-200*x+190.00)^2*exp
                (-100*(x-.95)^2-10*(y-1)^2)-4*cos(2*(y-1)*(x-.5)*pi)*(x-.5)^2*pi
                ^2*(3*exp(-100*(x-.95)^2-10*(y-1)^2)+1)-12*sin(2*(y-1)*(x-.5)*pi)*(x
                -.5)*pi*(-20*y+20)*exp(-100*(x-.95)^2-10*(y-1)^2)+3*cos(2*(y-1)*(x
                -.5)*pi)*(-20*y+20)^2*exp(-100*(x-.95)^2-10*(y-1)^2);
        end
    end
end
```

```
function [out] = x_refine_function(in)
   out=in;
end
```

```
function [out] = y_refine_function(in)
   out=in;
end
```

B Mesh Refinement Functions for the Run Example of Section 3.2.2

```
function [out] = x_refine_function(in)
  out=in;
  A=in(1);
  B=in(end);
  out=in.^2/B;
end
```

```
function [out] = y_refine_function(in)
  half=floor(length(in)/2)+1;
  out=in;
  A=in(1);
  B=in(half);
  out(1:half)=(4/(B-A)^3)*((in(1:half)-(B+A)/2).^3+((B-A)/2)^3)/2;
  A=in(half);
  B=in(end);
  out(half:end)=(4/(B-A)^3)*((in(half:end)-(B+A)/2).^3+((B-A)/2)^3)/2+0.5;
end
```

C Function Code for the Run Example of Section 4.2.2

```
function [out]=dirichlet_boundary(in)

epsilon=0.05;
out=zeros(1,length(in));
    for i=1:1:length(in)
        if (((in(i) < 0+epsilon)||(in(i) > 2*pi-epsilon))||
           ((in(i) < 3/7*pi+epsilon)&&(in(i) > 3/7*pi-epsilon))||
           ((in(i) < 25/42*pi+epsilon)&&(in(i) > 25/42*pi-epsilon))||
           ((in(i) < 3/4*pi+epsilon)&&(in(i) > 3/4*pi-epsilon))||
           ((in(i) < pi+epsilon)&&(in(i) > pi-epsilon))||
           ((in(i) < 3/2*pi+epsilon)&&(in(i) > 3/2*pi-epsilon))||
           ((in(i) < 13/8*pi+epsilon)&&(in(i) > 13/8*pi-epsilon)) )
             out(i)=0;
        else
             out(i)=NaN;
        end
    end
end
```

```
function [out]=neumann_boundary(in)
    for k=1:1:length(in)
          out(k)=0;
    end
end
```

```
function [out] = r_refine_function(in)
    B=in(end);
    out=-2*(0.5/B*in.^2-in);
end
```

```
function [out] = phi_refine_function(in)
    A=in(1);
    B=in(end);
    out=in;
end
```

```
function [f_mat]=forcing(phis,rs)
     l_r=length(rs);
     l_phi=length(phis);
     f_mat=zeros(l_phi,l_r);
     for k=1:1:l_r
          r=rs(k);
          for i=1:1:l_phi
               phi=phis(i);
               f_mat(i,k)=-1;
          end
     end
end
```

D Function Code for the Run Example of Section 5.2

```
function [diriBC]=dirichlet_boundary(phi,theta)
     N=length(phi);
     M=length(theta);
     diriBC=zeros(N,M);

     root1=pi/3;  root2=pi;
     epsilon=0.1;
     for j=1:1:M
          for i=1:1:N

               if ((theta(j)<pi/2+epsilon)&&(theta(j)>pi/2-epsilon)&&((phi(i)<root1+
                    epsilon)&&(phi(i)>root1-epsilon)))
                    diriBC(i,j)=1;
               elseif ((theta(j)<pi/2+epsilon)&&(theta(j)>pi/2-epsilon)&&((phi(i)<root2
                   +epsilon)&&(phi(i)>root2-epsilon)))
                    diriBC(i,j)=-10;
               else
                    diriBC(i,j)=NaN;
               end
          end
     end
end
```

```
function [neuBC]=neumann_boundary(phi,theta)
    N=length(phi);
    M=length(theta);
    neuBC=zeros(N,M);

    for j=1:1:M
        for i=1:1:N
            neuBC(i,j)=0;

        end
    end
end
```

```
function [out] = r_refine_function(in)
   B=in(end);
   out=-2*(0.5/B*in.^2-in);
end
```

```
function [out] = theta_refine_function(in)
   A=in(1);
   B=in(end);
   out=pi*(4/(B-A)^3)*((in-(B+A)/2).^3+((B-A)/2)^3);
end
```

```
function [out] = phi_refine_function(in)
   out=1/3*sin(3*in)+in;
end
```

```
function [f_mat]=forcing(phi,theta,r)
    l_r=length(r);
    l_theta=length(theta);
    l_phi=length(phi);
    f_mat=zeros(l_phi,l_theta,l_r);

    for k=1:1:l_r
        for j=1:1:l_theta
            for i=1:1:l_phi
                f_mat(i,j,k)=0;
            end
        end
    end

end
```