# Implementation of Laplace equation
using
# PETSc

SRIKANTH CHOWDADENAHALLI SATHYANARAYANA

# Contents

# 1

# Introduction

This report provides a brief introduction and usage of PETSc. Portable, Extensible Toolkit for Scientific Computation (PETSc) is a library containing routines useful in the domain of Scientific computing. It uses Message passing interface (MPI) for parallel communications. PETSc contains several libraries(1.1) which manipulate a set of objects(Vectors,Matrices etc.,). A large number of customizable linear and non linear solvers, Pre-conditioners, time steppers are available at hand to ease the process of scientific code development.

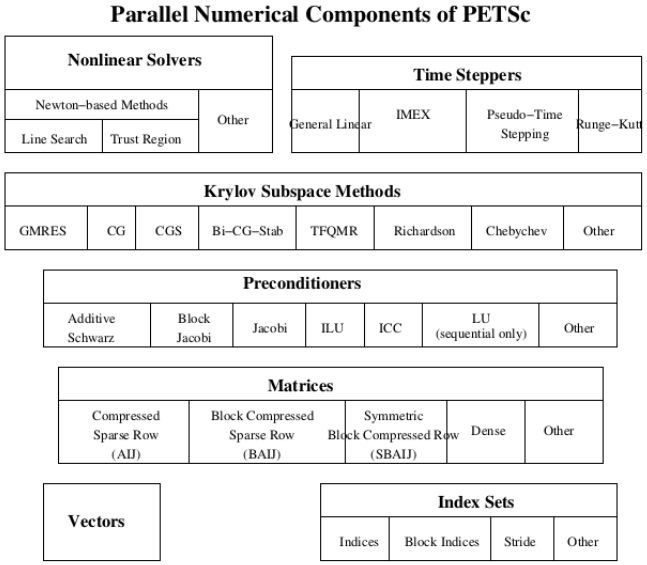


**Figure 1.1:** Numerical libraries of PETSc [1]

The succeeding content will describe the implementation of PETSc on a 2D Laplace equation. Further, the code with its accompanying terminologies is briefly explained.

## 1.1 Laplace equation

The Laplace equation in 2D solved in this code is given below,

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1.1}$$

$T$ is any physical variable like Temperature etc., and $x$ and $t$ represent the spatial coordinates.

## 1.2 Numerical Method

The governing equation is discretised using finite difference method and the final equation is expressed below.

A standard 5 point stencil is represented by,

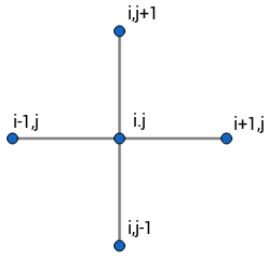


**Figure 1.2:** five point stencil

Using the above representation, the final equation can be written as,

$$T_{i,j} = \frac{1}{4}(T_{i,j+1} + T_{i,j-1} + T_{i,j+1} + T_{i,j-1}) \tag{1.2}$$

This equation is solved with the help of PETSc routines in Fortran programming language. A standard square domain with Dirichlet boundary condition is chosen and Gauss-Seidel iteration method is used for solving.

# 2

# Implementation

This section provides a brief description of the 2D Laplace code. The code begins with including necessary files for this problems. For example, Line one includes the PETSc routines.

```
#include <petsc/finclude/petscsys.h>
#include <petsc/finclude/petscis.h>
#include <petsc/finclude/petscvec.h>
#include <petsc/finclude/petscdm.h>
#include <petsc/finclude/petscdmda.h>
```

This code uses the Distributed arrays(DMDA) which implements the basic PETSc vector routine for regular rectangular grids. More information is provided in [1]. The basic varibale declaration is given below. Here, da is used for the distributed array.

```
      MPI_Comm comm
      Vec        X,Y,F,localX,Fold,Fnew
      DM         da
      PetscInt   Nx,Ny,N,mx,my,ifive,ithree
      PetscBool  flg,nooutput
      PetscReal    tol,error
      PetscReal    fnorm,fnorm1
      PetscInt             one
      PetscInt             ncount
      PetscScalar          mone
      PetscErrorCode ierr
```

Every PETSc implemented code starts with,

```
      call PetscInitialize(PETSC_NULL_CHARACTER,ierr)
```

And ends with,

```
      call PetscFinalize(ierr)
```

The above routine also initializes MPI. Every PETSc routine returns an integer which indicates whether the code contains an error. This check(in this case, ierr) must be added in all the routines.

The following code inputs the number of grids required in the x and y direction. The PetscOptionsGetInt routine is used to change values during run-time in the terminal.

```
      mx = 10
      my = 10
       call PetscOptionsGetInt(PETSC_NULL_OBJECT,PETSC_NULL_CHARACTER,      &
     &                         '-mx',mx,flg,ierr)
       call PetscOptionsGetInt(PETSC_NULL_OBJECT,PETSC_NULL_CHARACTER,      &
     &                         '-my',my,flg,ierr)
```

The Distributed array for to manage parallel grid and vector is initialized through,

```
      Nx = PETSC_DECIDE
      Ny = PETSC_DECIDE
       call DMDACreate2d(comm,DM_BOUNDARY_NONE,DM_BOUNDARY_NONE,            &
     &      DMDA_STENCIL_STAR,mx,my,Nx,Ny,one,one,                          &
     &      PETSC_NULL_INTEGER,PETSC_NULL_INTEGER,da,ierr)
```

The above grid creates a 2D grid in parallel. DM_BOUNDARY_NONE represents the non usage of ghost nodes. DMDA_STENCIL_STAR represents a stencil arrangement without the corner nodes. Nx,Ny allow Petsc to decide how the partitioning of the nodes is performed globally. The next input is the degree of freedom(one in this case). Further, the stencil width(one) is input. The stencil width one equates to a standard 5-point stencil. The next two inputs represent the arrays containing the number of nodes in each direction(this input is optional). the next value stores the distributed array structure(da).

```
      call DMCreateGlobalVector(da,X,ierr)
      call DMCreateLocalVector(da,localX,ierr)
```

DMCreateGlobalVector creates a global vector X without considering the ghost nodes. DMCreateLocalVector creates a local vector localX with ghost nodes.

```
      call FormInitialGuess(X,ierr)
      call ComputeFunction(X,F,ierr)
```

The above mentioned subroutines were created for this problem. FormInitialGuess takes the global vector, assigns the boundary conditions and initial guess in the domain. The implementation is provided in the code. ComputeFunction computes the initialized X for the discretized Laplace equation(1.2) and produces the solution vector F. More details on this implementation can be understood through the code.

```
      call DMDestroy(da,ierr)
      call VecDestroy(localX,ierr)
      call VecDestroy(X,ierr)
```

All PETSc objects must be destroyed in the end of the program.

# 3

# Directions to use the code

## 3.1 General instructions

The following points directs the user on how to compile and use the code

- Clone the code from Github.
- In the terminal, enter 'make laplace'.
- once the executable is generated type './laplace' to run the code.
- The code will output the number of iterations and final error.
- The user can run the code with a different grid size.
- The code can be run in parallel(for example, 2 processors) using 'mpiexec -n 2 ./laplace'.

## 3.2 General PETSc guidelines

- The common mistakes arise in not initializing PETSc, missing Fortran spacing to name a few.
- Always refer the manual before implementing your own code.
- Use the PETSc viewing options to understand the numbering used and to check for errors.
- It is recommended to use Valgrind to check for memory leaks in case of any errors.
- To understand the fundamental concepts of PETSc and its implementation, go through the tutorials.

# Bibliography

[1] PETSc Users Manual(Revision 3.7), Mathematics and Computer Science Division, Argonne National Laboratory.