ME630 Assignment 2

Aman Parekh (180073)

September 22, 2021

$$\boxed{\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = g(x, y)}$$

For the above elliptic equation, it is discretized in the following manner and different methods are used to solve it:

$$\phi(i,j) = \frac{g(i,j) - \left\{ \frac{1}{\Delta y^2}\phi(i,j-1) + \frac{1}{\Delta x^2}\phi(i-1,j) + \frac{1}{\Delta x^2}\phi(i+1,j) + \frac{1}{\Delta y^2}\phi(i,j+1) \right\}}{-2\left\{ \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \right\}}$$

The code for all five methods is attached with the submission and the following commands can be used to run the cases for each method:

1. **Jacobi:**

```
>> f95 -o jacobi meshgen.f90 functions.f90 main_jacobi.f90 updatebc.f90
>> ./jacobi
```

2. **Gauss-Siedel:**

```
>> f95 -o siedel meshgen.f90 functions.f90 main_siedel.f90 updatebc.f90
>> ./siedel
```

3. **SOR($\lambda = 1.2$):**

```
>> f95 -o sor meshgen.f90 functions.f90 main_sor.f90 updatebc.f90
>> ./sor
```

4. **RBPGS:**

```
>> f95 -o rbpgs meshgen.f90 functions.f90 main_rbpgs.f90 updatebc.f90
>> ./rbpgs
```

5. **ADI:**

```
>> f95 -o jacobi meshgen.f90 functions.f90 main_adi.f90 updatebc.f90 tdma.f90
>> ./adi
```

Each code generates 4 files for each case: i) `convergence.dat` - Stores 2 columns of iterations and the corresponding residual, ii) `numerical.dat`, iii) `analytical.dat` and iv) `error.dat` which store the value at each node of the domain, with the the first row being the top of the domain and last row being the bottom of the domain.

Residual vs Iterations

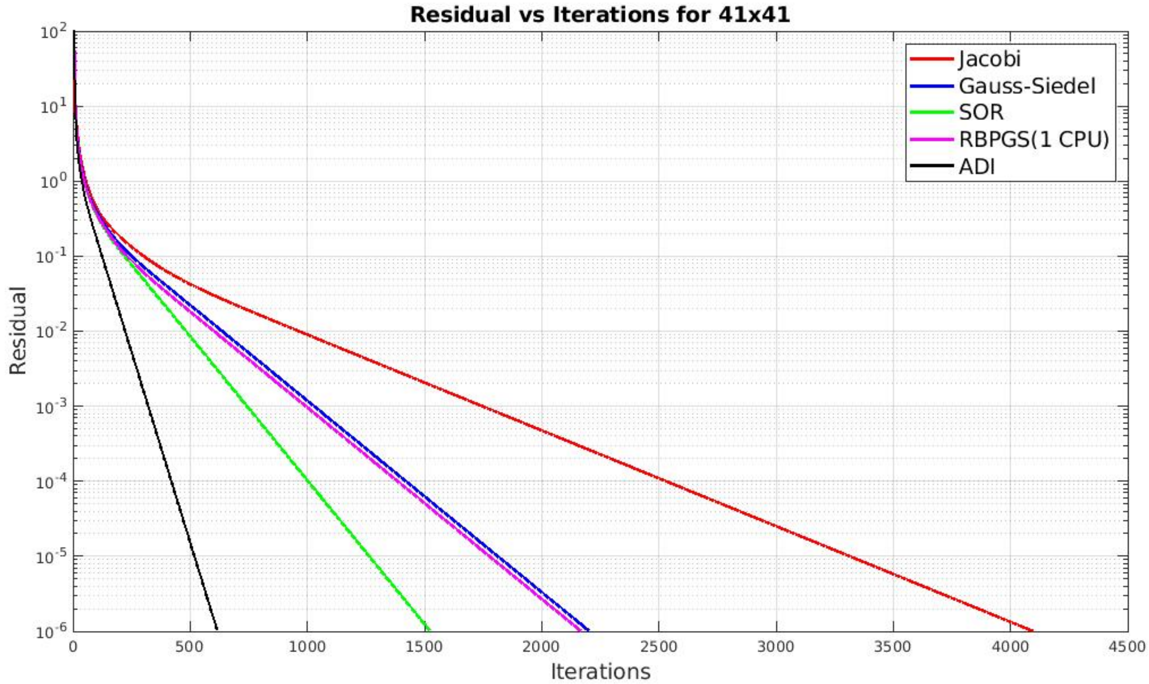

Figure 1: Residual vs Iterations for 41x41

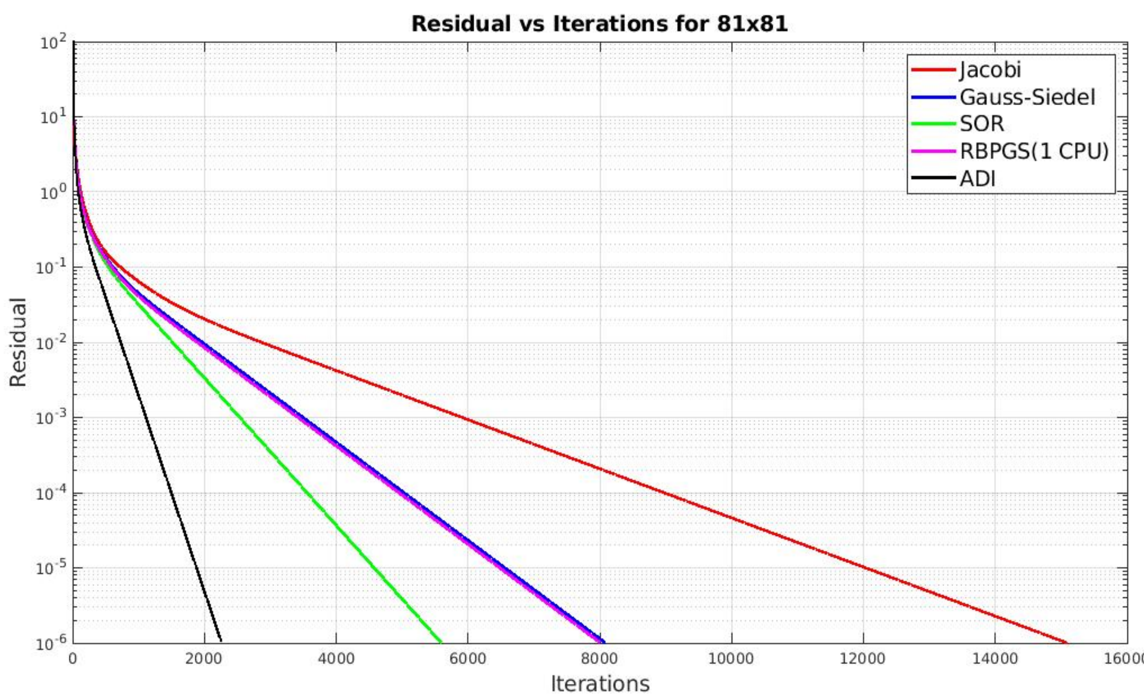

Figure 2: Residual vs Iterations for 81x81

The following are the observations:

1. Clearly, ADI is the fatest method to arrive at convergence and the complete order is:

$$\text{ADI} > \text{SOR} > \text{RBPGS} > \text{Gauss-Siedel} > \text{Jacobi}$$

2. The RBPGS algorithm should result in faster convergence but since we use only 1 CPU in the current simulations, the time taken by it is very similar to Gauss-Siedel. This is because the RBPGS is formulated in the same way as Gauss-Siedel, except alternating points are calculated in two separate loops.

3. ADI is fastest because in that method we solve each row and column at once implicitly, so the boundary condition travels faster into the domain.

4. Gauss-Siedel, SOR and RBPGS are faster than Jacobi because in those methods we use the already updated values of the ϕ matrix to calculate new values, but in Jacobi, we use the old ϕ to calculate to all new values.

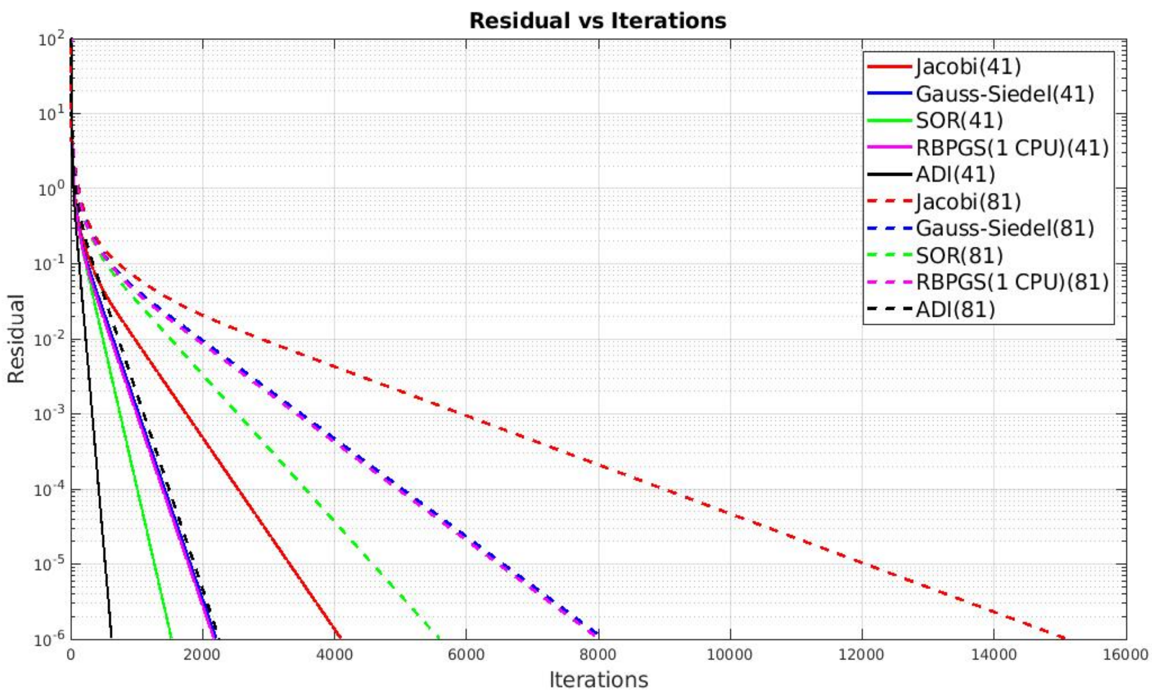

Figure 3: Residual vs Iterations

Observations in the following plots:

1. The numerical solution agrees well with the analytical solution in all cases.

2. All the solutions are almost identical for all the methods applied, in both mesh distributions. The difference is in the number of iterations required.

3. The maximum error is seen in the regions near (0,0) and (1,1). This could be because the analytical solution sees maximum gradient in those regions as well.

4. However, for the 41x41 grid, the |error| = 0.2, whereas in 81x81, the |error| = 0.06. Clearly, the error has reduced on increasing the mesh resolution.

Note: Another, rather weird, observation was that if the boundary condition update was kept after the main loop, the solution did not seem to converge below 10^{-6}. However, in the current version, the boundary condition update is kept before the main loop and they converge really well for all methods. I was not able to point out the cause of this behaviour.

Plots for 41x41 Mesh

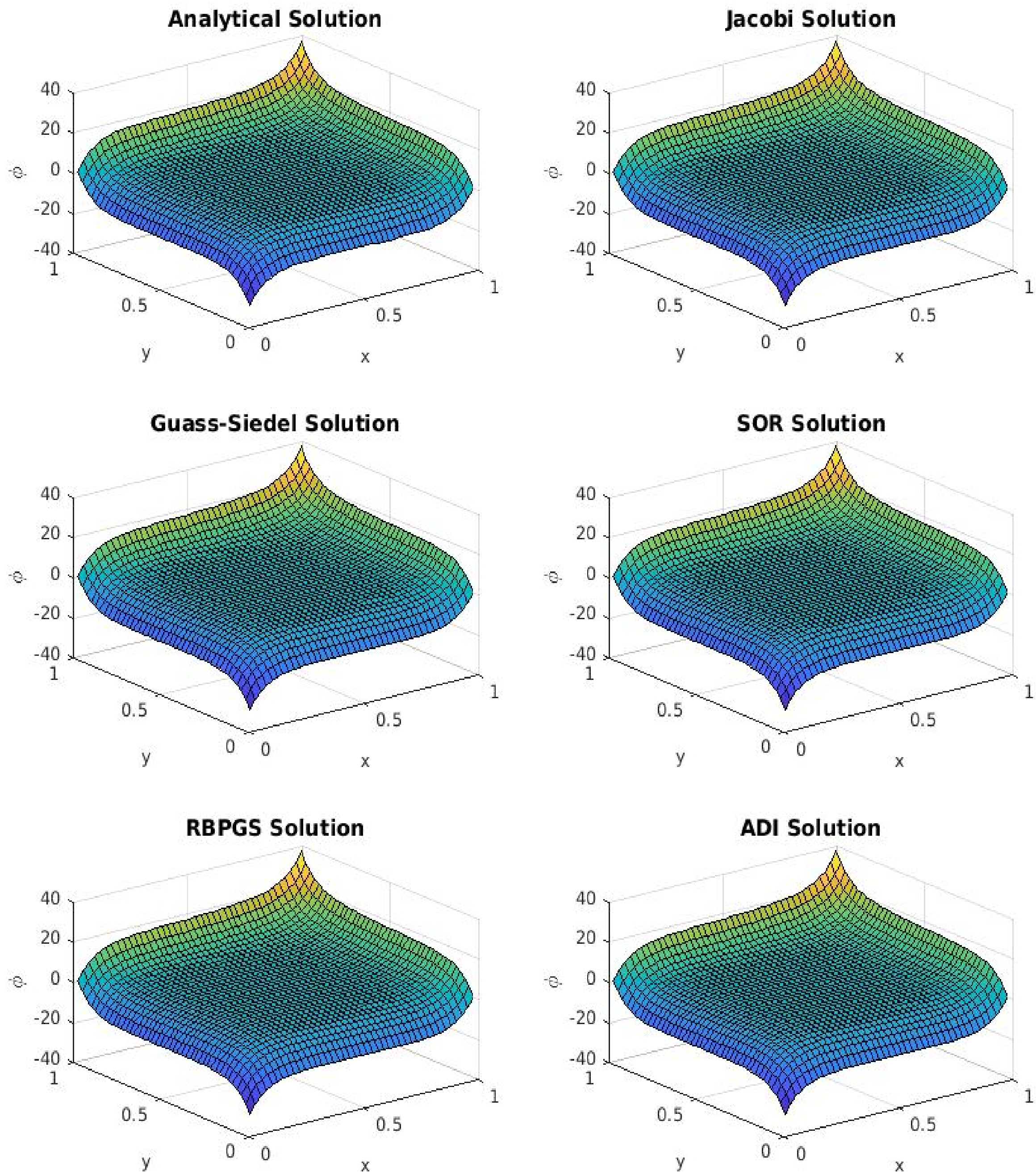

Figure 4: Analytical and Numerical Solution for 41x41

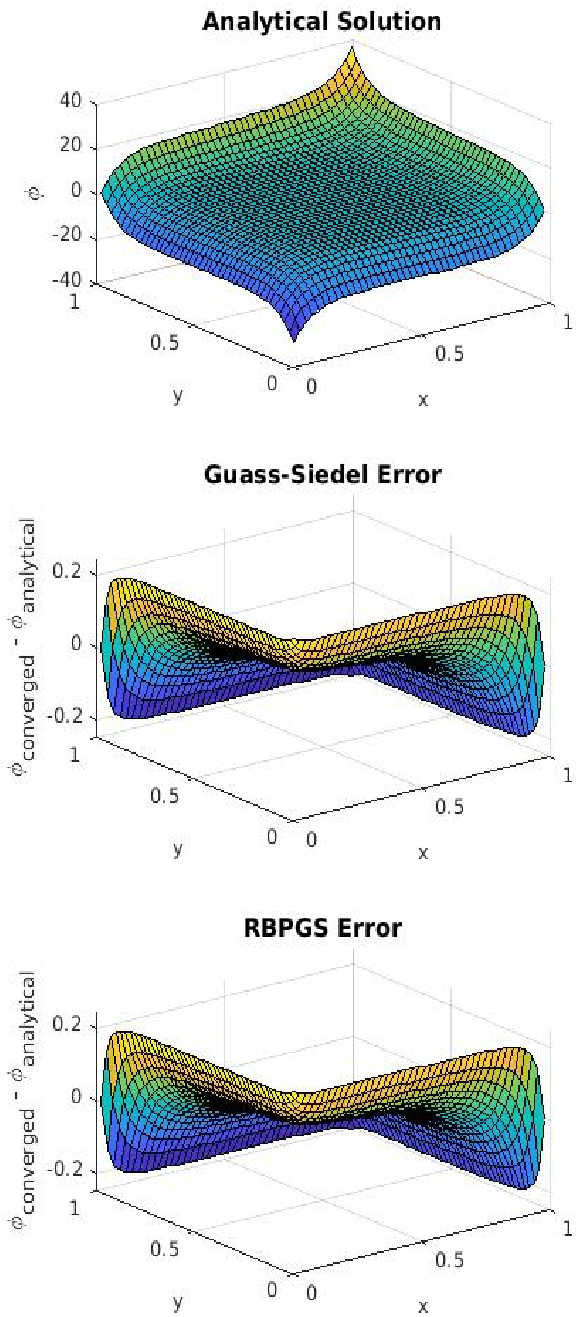

Figure 5: Error for 41x41

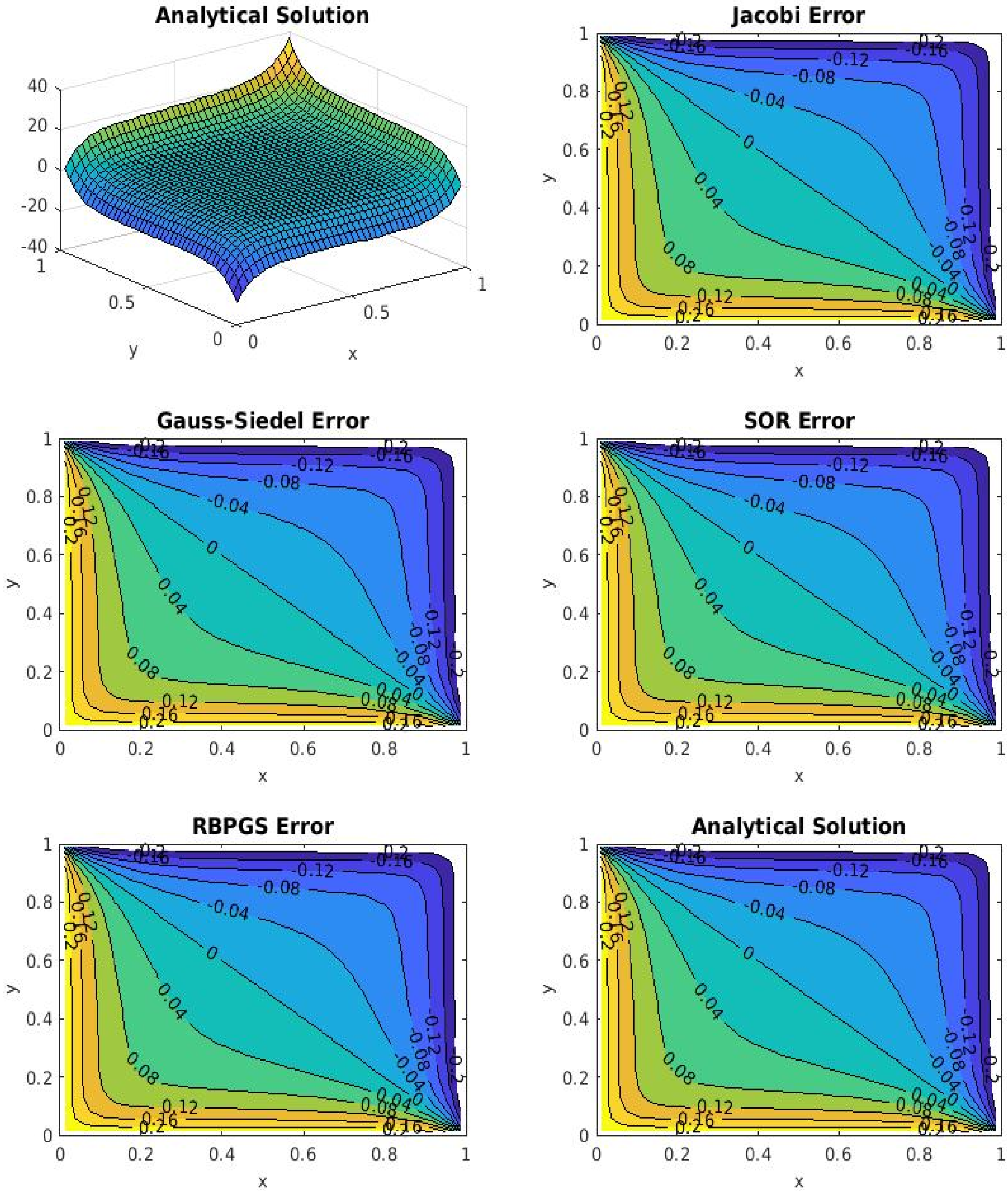

Figure 6: Error for 41x41 (2D)

Plots for 81x81 Mesh

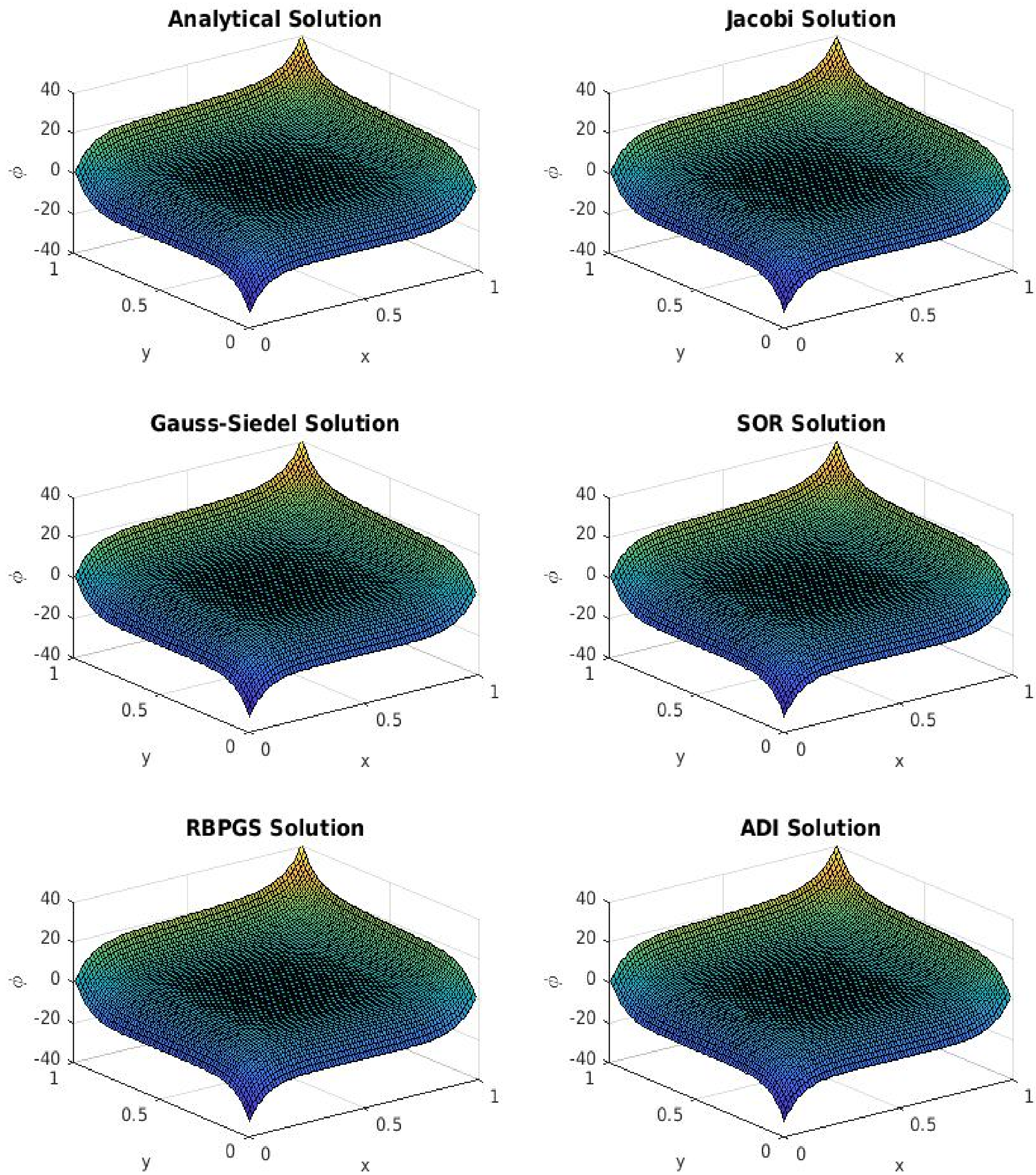

Figure 7: Analytical and Numerical Solution for 81x81

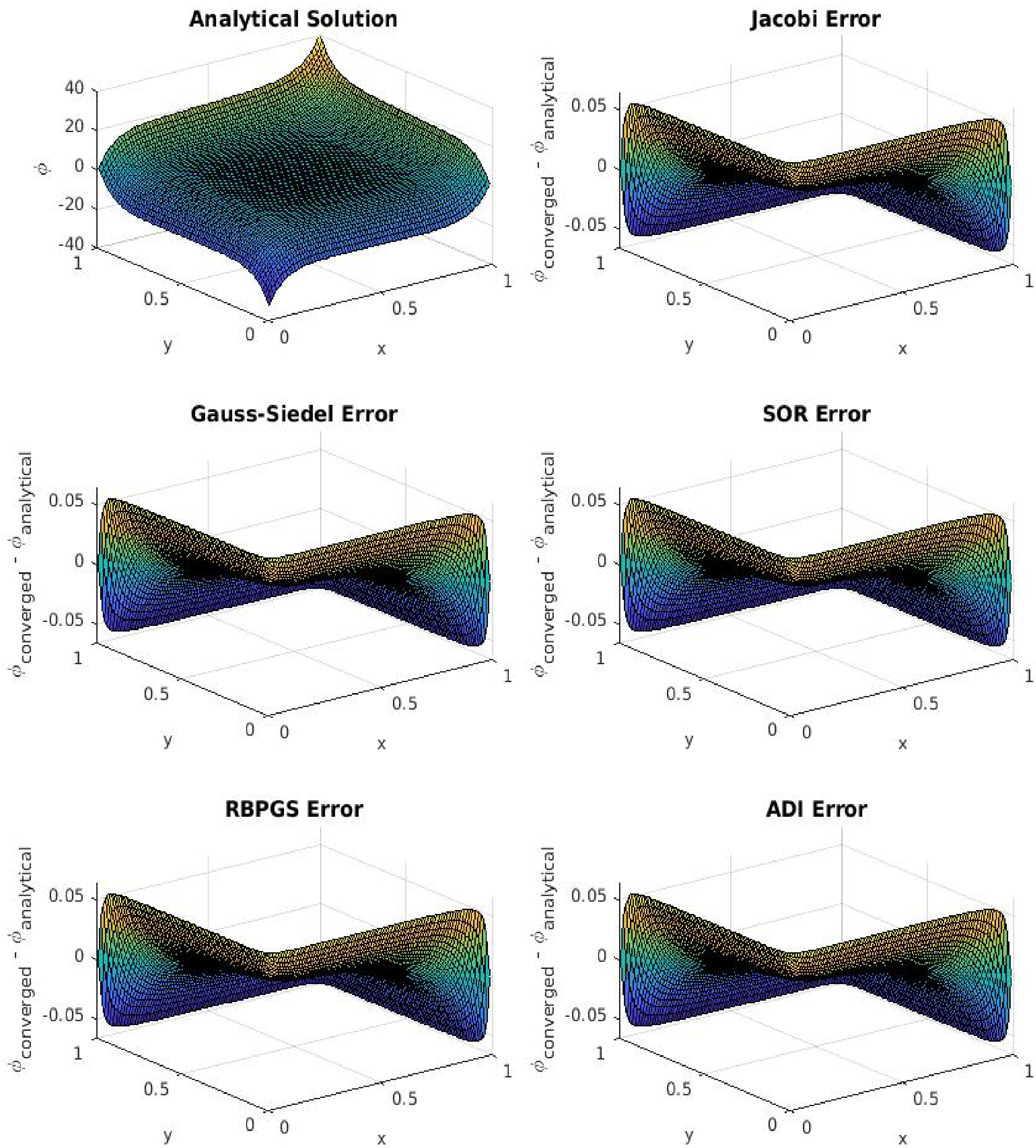

Figure 8: Error for 81x81

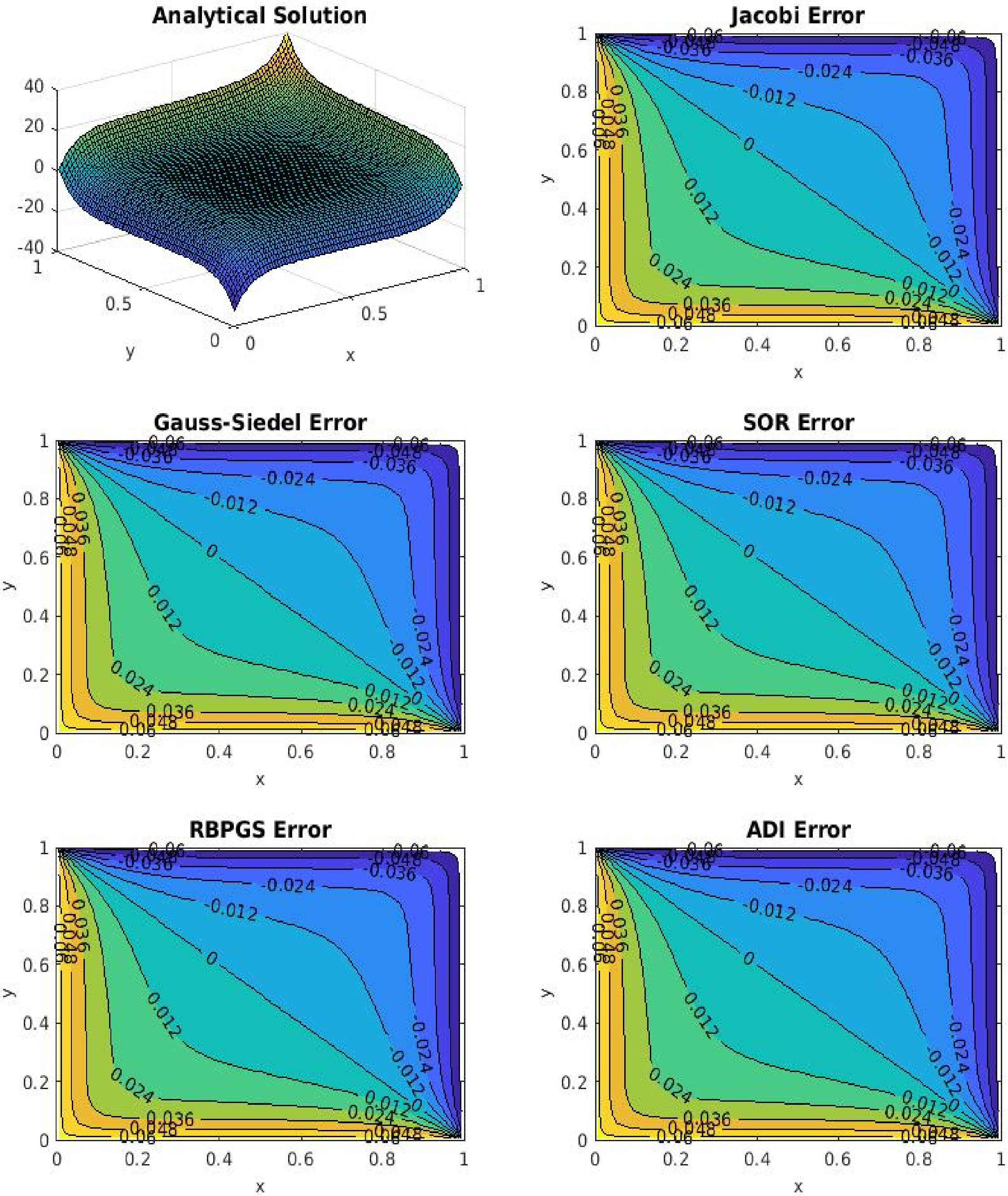

Figure 9: Error for 81x81 (2D)

FORTRAN Codes

The following are the codes that are common to all the methods, which are: subroutine for mesh generation, functions for calculating source term, analytical solution and residual and the subroutine for boundary condition update. The code specific for each method is included in the submission.

1. **subroutine meshgen**

```
! AMAN PAREKH - 180073 - Fall 2021 - ME630

subroutine meshgen(n, mesh)     ! Subroutine for Mesh Generation

      implicit none
      integer :: n, i, j, c
      real*8 :: mesh(4,n*n)
      real*8 :: edge1, edge2, total_length, del

      edge1 = -0.5/(n-2.0)
      edge2 = 1.0 + (0.5/(n-2.0))
      total_length = edge2 - edge1
      del = (total_length*1.0)/(n-1.0)

      ! mesh = [x_index y_index x_loc y_loc]

      c = 1
      do j = 1,n
        do i = 1,n
          mesh(1,c) = i
          mesh(2,c) = j
          mesh(3,c) = edge1 + ((i-1)*1.0)*del
          mesh(4,c) = edge1 + ((j-1)*1.0)*del
          c = c + 1
        end do
      end do

end subroutine meshgen
```

---

2. **functions**

```
! AMAN PAREKH - 180073 - Fall 2021 - ME630

real*8 function g(x,y)         ! Function for Source Term

        implicit none

        real*8 :: x,y

        g = 2.0*sinh(10.0*(x-0.5)) + 40.0*(x-0.5)*cosh(10.0*(x-0.5))  &
           + 100.0*((x-0.5)**2)*sinh(10.0*(x-0.5))                    &
           + 2.0*sinh(10.0*(y-0.5)) + 40.0*(y-0.5)*cosh(10.0*(y-0.5)) &
           + 100.0*((y-0.5)**2)*sinh(10.0*(y-0.5))                    &
           + 4.0*(x**2 +y**2)*exp(2.0*x*y)

end function g

real*8 function analytical(x,y)        ! Calculates Analytical Solution

        implicit none

        real*8 :: x,y
```

```fortran
        analytical = ((x-0.5)**2)*sinh(10.0*(x-0.5)) + ((y-0.5)**2)*sinh(10.0*(y-0.5)) + exp(2*x*y)

end function analytical

real*8 function residual(n, phi, phi_old)

        implicit none

        integer :: n, i, j
        real*8 :: phi(n,n), phi_old(n,n), s

        s = 0
        do j = 2,n-1
          do i = 2,n-1
            s = s + (phi(i,j) - phi_old(i,j))**2
          end do
        end do

        residual = sqrt(s)

end function residual
```

3. **subroutine updatebc**

```fortran
! AMAN PAREKH - 180073 - Fall 2021 - ME630

subroutine updatebc(n, phi, mesh)

        implicit none

        integer :: n, i, j
        real*8 :: x, y
        real*8 :: phi(n,n), mesh(4,n*n)

        ! Left Boundary
        do j = 2,n-1
            y = mesh(4,(j-1)*n + 1)
            phi(1,j) = 2.0*(0.25*sinh(-5.0) + ((y-0.5)**2)*sinh(10.0*(y-0.5)) + 1) - phi(2,j)
        end do

        ! Right Boundary
        do j = 2,n-1
            y = mesh(4,(j)*n - 1)
            phi(n,j) = 2.0*(0.25*sinh(5.0) + ((y-0.5)**2)*sinh(10.0*(y-0.5)) + exp(2.0*y)) - phi(n-1,j)
        end do

        ! Bottom Boundary
        do i = 2,n-1
            x = mesh(3,i)
            phi(i,1) = 2.0*(0.25*sinh(-5.0) + ((x-0.5)**2)*sinh(10.0*(x-0.5)) + 1) - phi(i,2)
        end do

        ! Top Boundary
        do i = 2,n-1
            x = mesh(3,(n-1)*n + i)
            phi(i,n) = 2.0*(0.25*sinh(5.0) + ((x-0.5)**2)*sinh(10.0*(x-0.5)) + exp(2.0*x)) - phi(i,n-1)
        end do

end subroutine updatebc
```