1 Introduction

1.1 Motvation

We first encountered Laplace Equation during our course in electricity and magnetism in second semester and we were fascinated with how one can calculatge the potential in a region just by knowing the boundary condition, ofcourse the region has to be charge free for applying Laplace Equation. After Laplace Equation , we were introduced to Poisson Equation which was able solve in region having charges(or sources). When we were given the oppurtunity to choose a project in our computational physics charge this semester, it did not take us long to decide the topic for project.

1.2 General Idea

In our project we will try to tackle the Laplace and Poisson equaton which is an ellipitic linear partial differential equation having application in various fields of physics ranging from thermodynamics, electrostatics etc. We will solve the equation computationally using the method of finite differences in one and two dimensions for rectangular membrane

2 Algprithm

Data: INPUT -: An $k \times m$ matrix of initial values, value of step size h and also a matrix containing the initial charge configuration

Result: OUTPUT -: An $k \times m$ matrix containing the values of potential on all x and y values

```
for f = 0, 1, 2, 3, 4....N do
    make new array of size k × m ;          /* initialising a new array for solution */
    for i =0, 1, 2, 3 ... k do ;            /* taking a x value */

        for j =1, 2, 3 ... (m-1) do ;       /* taking all y value for a x value */

            /* now defining the required quantites for stencil */
            left = a_{i,j-1} ;
            right = a_{i,j+1} ;
            if i = k - 1 then
                up = a_{i-1,j};
                else
                    up = a_{i+1,j};
                end
            end
            if i = 0 then
                down = a_{i+1,j};
                else
                    down = a_{i-1,j} ;
                end
            end
            new a_{i,j} = (up + down + left + right + h^2 * p_{i,j} )/4 + new a_{i,j} ;   /* new value the grid point */
        end
        max relative error = max(new x - x)/ x;
        if max relative error < tolerance then ;   /* checking for tolerance */

            break;
            else ;          /* if tolerance not reached then the iteration continues */

                new x = x
            end
        end
    end
end
```

Algorithm 1: Jacobi Method

Data: INPUT -: An $k \times m$ matrix of initial values, value of step size h, value of ω or relaxation factor and also a matrix containing the initial charge configuration

Result: OUTPUT -: An $k \times m$ matrix containing the values of potential on all x and y values

```
for f = 0, 1, 2, 3, 4....N do
    make new array of size k × m ;          /* initialising a new array for solution */
    for i =0, 1, 2, 3 ... k do ;                                   /* taking a x value */

        for j =1, 2, 3 ... (m-1) do ;                 /* taking all y value for a x value */

            /* now defining the required quantites for stencil                          */
            left = a_{i,j-1} ;
            right = a_{i,j+1} ;
            if i = k - 1 then
                up = a_{i-1,j};
                else
                    up = a_{i+1,j};
                end
            end
            if i = 0 then
                down = a_{i+1,j};
                else
                    down = a_{i-1,j} ;
                end
            end
            new a_{i,j} = ((up + down + left + right + h^2 * p_{i,j} )/4 -new a_{i,j})*omega + new a_{i,j} ;      /* new
            value the grid point */
        end
        max relative error = max((new x - x)/ x);
        if max relative error < tolerance then ;                  /* checking for tolerance */

            break;
            else ;        /* if tolerance not reached then the iteration continues */

                new x = x
            end
        end
    end
end
```

Algorithm 2: Successive Over Relaxation

```
Data: INPUT -: An k × m matrix of initial values, value of step size h and also a matrix containing the
      initial charge configuration
Result: OUTPUT -: An k × m matrix containing the values of potential on all x and y values
for f = 0, 1, 2, 3, 4....N do
    make new array of size k × m ;          /* initialising a new array for solution */
    for i =0, 1, 2, 3 ... k do ;                               /* taking a x value */

        for j =1, 2, 3 ... (m-1) do ;          /* taking all y value for a x value */

            /* now defining the required quantites for stencil                    */
            left = a_{i,j-1} ;
            right = a_{i,j+1} ;
            if i = k - 1 then
                up = a_{i-1,j};
                else
                    up = a_{i+1,j};
                end
            end
            if  i = 0 then
                down = a_{i+1,j};
                else
                    down = a_{i-1,j} ;
                end
            end
            new a_{i,j} = ((up + down + left + right + h^2 * p_{i,j} )/4 -new a_{i,j}) + new a_{i,j} ; /* new value the
            grid point */
        end
        max relative error = max((new x - x)/ x);
        if max relative error < tolerance then ;            /* checking for tolerance */

            break;
            else ;       /* if tolerance not reached then the iteration continues */

                new x = x
            end
        end
    end
end
```

Algorithm 3: Gauss Seidel Method