FLORIDA INSTITUTE OF TECHNOLOGY
MECHANICAL AND AEROSPACE ENGINEERING DEPARTMENT

MAE 5150-E1: Computational Fluid Dynamics

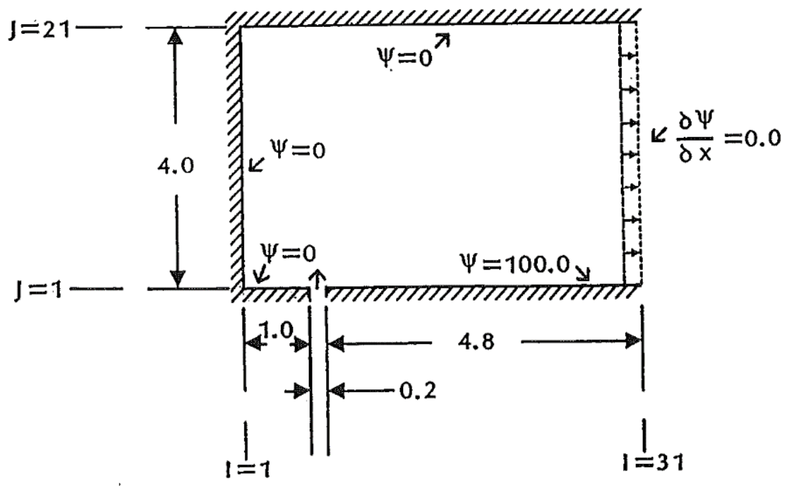
Fall 2017

Coding Project 2

Name: Max Le

Due October 17, 2017

A two-dimensional inviscid, incompressible fluid is flowing steadily through a chamber between the inlet and the outlet, as shown in the figure. It is required to determine the streamline pattern within the chamber.



For a two-dimensional, incompressible flow, the continuity equation is expressed as

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

A stream function Ψ may be defined such that

$$u = \frac{\partial \Psi}{\partial y} \quad \text{and} \quad v = -\frac{\partial \Psi}{\partial x}$$

Recall that a streamline is a line of constant stream function. Furthermore, vorticity is defined as

$$\Omega = \nabla \times V$$

for which

$$\Omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

For an irrotational flow, the vorticity is zero. Therefore,

$$\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = 0$$

Substituting the definitions of the stream function into the above equations yields

$$\frac{\partial}{\partial x}\left(-\frac{\partial \Psi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial \Psi}{\partial y}\right) = 0$$

or

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = 0$$

The goal in this problem is to obtain the solution of this elliptic partial differential equation using the various numerical techniques discussed earlier. The solution will provide the streamline pattern within the chamber.

Since the chamber walls are streamlines, i.e. lines of constant Ψ, we will assign values for these streamlines, as shown in the figure. Solve this problem with codes using the following techniques:

a) Point Gauss-Seidel	c) Point SOR
b) Line Gauss-Seidel	d) Line SOR

For all methods, the step sizes are specified as

$$\Delta x = 0.2, \qquad \Delta y = 0.2 \qquad ERRORMAX = 0.01$$

with convergence criterion *ERROR* < *ERRORMAX*, where

$$ERROR = \sum_{\substack{i=2\\ j=2}}^{\substack{j=JM-1\\ i=IM-1}} \left| \Psi_{i,j}^{k+1} - \Psi_{i,j}^{k} \right|$$

Use an initial data distribution of $\Psi = 0.0$. Plot the streamline pattern (lines of constant Ψ). Rerun the SOR codes for several values of the relaxation parameter and plot the relaxation parameter versus the number of iterations for these two schemes. In each case, determine the optimal value of the relaxation parameter.

Submit a both a hardcopy of your code and an electronic copy on Canvas.

Note: Except for the purpose of creating plots, MATLAB, Excel, or other commercially available software may not be used for this assignment. You may otherwise program in any language you wish.

```
//MAE 5150: Coding Project 2
//Max Le
//MAIN PROGRAM
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include "CFD_PROJECT2_HEADERS.h"
#include <ctime>
using namespace std;
float wi;//dummy variable to test W later on

int main(){

    //CALL POINT GAUSS SEIDEL
    Elliptic PGS;
    InitializePsi(&PGS,wi);
    PointGaussSeidel(&PGS);

    //CALL LINE GAUSS SEIDEL
    Elliptic LGS;
    InitializePsi(&LGS,wi);
    LineGaussSeidel(&LGS);

    // //CALL POINT SOR AT OPTIMUM VALUE OF W
    // {
    //  float wi = 1.801;
    //  Elliptic PSOR;
    //  InitializePsi(&PSOR,wi);
    //  PointSOR(&PSOR);
    // }

    // //CALL LINE SOR AT OPTIMUM VALUE OF W
    // {
    //  float wi = 1.3;
    //  Elliptic LSOR;
    //  InitializePsi(&LSOR,wi);
    //  LineSOR(&LSOR);
    // }

    //CALL THIS FUNCTION TO SHOW TABLES OF W VS. ITERATION
    //IF CALL THIS FUNCTION, THEN COMMENT OUT THE INDIVIDUAL PSOR/LSOR AT
OPTIMUM W
    // PrintTablesAndCompare();
    return 0;
}
//END MAIN PROGRAM
```

```
//MAE 5150: Coding Project 2
//Max Le
//HEADER FILE WITH FUNCTIONS
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <ctime>
using namespace std;

//DECLARE STRUCTURE
struct Elliptic
{
    double dx ;
    double dy;
    double imax;
    double jmax;
    double ERROR_MAX;
    vector<vector <double> > psi;
    vector<vector <double> > psi_updated;
    double ERROR;
    int maxiter;
    int iter;
    double w ; //relaxation parameter
    double beta;
    double oneby2BetaSquare;
    double betaSquare;
    double psi1;
    double psi3;
    double OneMinusOmega;
    double OmegaBetaSquare;
    double TimesBetaSquare;
    double Omegaby2BetaSquare;
    float timeElapsed;
};

//FUNCTION TO RESET PSI EVERYTIME IT'S CALLED (BC, ZEROS)
void InitializePsi(struct Elliptic *laplace, float wi){
    laplace->dx = 0.2;
    laplace->dy = 0.2;
    laplace->imax = 31;
    laplace->jmax = 21;
    laplace->ERROR_MAX = 0.01; //max allowable error
    laplace->ERROR=0.0;
    laplace->maxiter = 500;
    laplace->iter = 0;
    laplace->psi.resize(laplace->jmax+1);
    laplace->psi_updated.resize(laplace->jmax+1);
    laplace->w = wi;
    laplace->beta = (laplace->dx)/(laplace->dy);
    laplace->oneby2BetaSquare = (1./(2*(1+pow(laplace->beta,2.))));
    laplace->betaSquare = pow(laplace->beta,2.);
    laplace->psi1 = 0.0;
    laplace->psi3 = 100.00;
    laplace->OneMinusOmega = (1-laplace->w);
    laplace->OmegaBetaSquare = laplace->w*laplace->betaSquare;
    laplace->TimesBetaSquare = 2.*(1+pow(laplace->beta,2.));
    laplace->Omegaby2BetaSquare = (laplace->w/(2.*(1+pow(laplace->beta,2.))));

    for (int j = 1; j<=laplace->jmax;j++){
        laplace->psi[j].resize(laplace->imax+1);
        laplace->psi_updated[j].resize(laplace->imax+1);
    }

    //SET zero to everywhere
    for (int j = 1; j<=laplace->jmax;j++){
        for (int i = 1; i<=laplace->imax;i++){
            laplace->psi[j][i] = 0.0;
            laplace->psi_updated[j][i] = 0.0;       }
    }
    // //bottom BC,initially ZERO everywhere
    for (int i = 7; i<=laplace->imax;i++){
        laplace->psi[1][i] = laplace->psi3;
        laplace->psi_updated[1][i] = laplace->psi3;
    }
}

//THOMAS ALGORITHM
void thomasTriDiagonalX(int j, double a[],double b[],double c[], double d[],
struct Elliptic *laplace){

    int imax = laplace->imax-1;
    double dprime[imax+1];
    double cprime[imax+1];
    dprime[1] = d[1];
    cprime[1] = c[1];

    //FORWARD LOOP
    for (int i = 2; i<=imax;i++){
        dprime[i] = d[i] - ((b[i]*a[i-1])/(dprime[i-1]));
        cprime[i] = c[i] - ((cprime[i-1]*b[i])/(dprime[i-1]));
    }

    laplace->psi_updated[j][imax] = cprime[imax]/dprime[imax];

    //BACKWARD LOOP
    for (int i = imax-1;i>=2;i--){
        laplace->psi_updated[j][i] = (cprime[i]-(a[i]*laplace->psi_updated[j][i
+1]))/(dprime[i]);
    }
}
```

```
//POINT GAUSS SEIDEL
void PointGaussSeidel(struct Elliptic *laplace){
    do{
        laplace->ERROR = 0.0;
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i = 2; i<=laplace->imax-1;i++){
                    //Finite Difference, using psi_updated as Latest Data

                    laplace->psi_updated[j][i] = (laplace->oneby2BetaSquare)*
(laplace->psi[j][i+1]+laplace->psi_updated[j][i-1]+(laplace->betaSquare)*
(laplace->psi[j+1][i]+laplace->psi_updated[j-1][i]));

                    //Calculate error, keep doing this until satisfy ERROR MAX
                    laplace->ERROR += abs((laplace->psi_updated[j][i] -
laplace->psi[j][i]));
            }
        }
        //Updating Pupdated with P
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i =2; i<=laplace->imax-1;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }
        // Make sure BC satisfied, dpsi/dx = 0
        for (int j = 2; j<=laplace->jmax-1;j++){
                laplace->psi[j][laplace->imax] = laplace->psi[j][laplace-
>imax-1];
        }
        //Update the iteration counter
        laplace->iter = laplace->iter + 1;
        }while(laplace->ERROR > laplace->ERROR_MAX);

        printf("Point GS Converged! Max iter is: %d \n", laplace->iter);

        // //PRINTING TO FILE

        FILE * outfile1;
        outfile1 = fopen("ResultsPointGS.dat","w");

        for (int j = 1; j<=laplace->jmax;j++){
            for (int i =1; i<=laplace->imax;i++){
                fprintf(outfile1,"%6.9f\t", laplace->psi[j][i]);
            }
            fprintf(outfile1,"\n");
        }
}

//LINE GAUSS SEIDEL
void LineGaussSeidel(struct Elliptic *laplace){

    //THOMAS PARAMETERS
    //Define vectors for Thomas (in X)
```

```
int imax = laplace->imax;
double ax[imax+1]; //above
double bx[imax+1]; //below
double cx[imax+1];//rhs
double diagonalX[imax+1];//diagonal
//Fill out zero values to all ax.bx.cx.Dx
for (int i = 0;i<=imax+1;i++){
    ax[i] = 0.0;
    bx[i] = 0.0;
    cx[i] = 0.0;
    diagonalX[i] = 0.0;
}
//Fill out values
//For ay,by,Dy from 1 to JMAX for now
//Rewrite value below
for (int i=1; i<=laplace->imax;i++){
    ax[i] = 1.;
    bx[i] = 1.;
    diagonalX[i] = -1.0*laplace->TimesBetaSquare;
}
//Special values rewrite
//Don't use zero elements; zero out these
//Really dont need, because loop at 2 to imax-1
ax[0] = 0.0;
bx[0] = 0.0;
cx[0] = 0.0;
diagonalX[0] = 0.0;
//All the 1st elements = 0
ax[1] = 0.0;
bx[1] = 0.0;
cx[1] = laplace->psi1; //which is 0
diagonalX[1] = 1.0;
//All last values = 0
ax[imax] = 0.0;
bx[imax] = 0.0;
cx[imax] = 0.0;
diagonalX[imax] = 1.0;
//Diagonals, d has full, a misses last, b misses first
ax[imax-1] = 0.0;// a misses last
bx[2] = 0.0; //b misses first

//LINE GAUSS SEIDEL LOOP
do{
    laplace->ERROR = 0.0;
    // Loop through jTH row
    for (int j = 2; j<=laplace->jmax-1;j++){
        //Loop through iTH row
        for (int i = 2; i<=laplace->imax-1;i++){
            // //BOUNDARY CONDITIONS
            //at I = 2, zero
            if (i == 2){
                    cx[i] = (
                        (-laplace->betaSquare*(laplace->psi[j+1][i]))+
                        (-laplace->betaSquare*laplace->psi_updated[j-1][i])-
                        (laplace->psi1)
                    );
                }
                //at I = IMAX-1, will have psi[IMAX], which needs dPsi/dx = 0
                else if (i == laplace->imax-1){
                    cx[i] = (
                        (-laplace->betaSquare*(laplace->psi[j+1][i]))+
                        (-laplace->betaSquare*laplace->psi_updated[j-1][i])-
                        (laplace->psi_updated[j][i+1])
                    );
                }
                else{
                    cx[i] = (
                        -(laplace->betaSquare*laplace->psi[j+1][i])-
                        (laplace->betaSquare*laplace->psi_updated[j-1]
[i]));
                }
            }
            thomasTriDiagonalX(j,ax,bx,cx,diagonalX,laplace);

            laplace->psi_updated[j][laplace->imax] =
                laplace->psi_updated[j][laplace->imax-1];

            for(int i=1; i<=laplace->imax; i++) {
                laplace->ERROR +=
                    abs((laplace->psi_updated[j][i] - laplace->psi[j][i]));
            }
        }
        //Updating Pupdated with P
        for (int j = 1; j<=laplace->jmax;j++){
            for (int i =1; i<=laplace->imax;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }
        //Update the iteration counter
        laplace->iter = laplace->iter + 1;

    }while(laplace->ERROR > laplace->ERROR_MAX);

    printf("Line GS Converged! Max iter is: %d\n", laplace->iter);

    //PRINTING TO FILE
    FILE * outfile2;
    outfile2 = fopen("ResultsLineGS.dat","w");
    for (int j = 1; j<=laplace->jmax;j++){
        for (int i =1; i<=laplace->imax;i++){
            fprintf(outfile2,"%6.9f\t", laplace->psi[j][i]);
        }
        fprintf(outfile2,"\n");
```

```c
    }
}


//LINE SOR
void LineSOR(struct Elliptic *laplace){

    //start clock
    clock_t t;

    t = clock();

    int imax = laplace->imax;
    double ax[imax+1]; //above
    double bx[imax+1]; //below
    double cx[imax+1];//rhs
    double diagonalX[imax+1];//diagonal

    //Fill out zero values to all ax.bx.cx.Dx

    for (int i = 0;i<=imax+1;i++){
        ax[i] = 0.0;
        bx[i] = 0.0;
        cx[i] = 0.0;
        diagonalX[i] = 0.0;
    }

    //Fill out values
    for (int i=1; i<=laplace->imax;i++){
        ax[i] = laplace->w;
        bx[i] = laplace->w;
        diagonalX[i] = -1.0*laplace->TimesBetaSquare;
    }

    //Special values rewrite
    //Don't use zero elements; zero out these
    //Really dont need, because loop at 2 to jmax-1
    ax[0] = 0.0;
    bx[0] = 0.0;
    cx[0] = 0.0;
    diagonalX[0] = 0.0;
    //All the 1st elements = 0
    ax[1] = 0.0;
    bx[1] = 0.0;
    cx[1] = laplace->psi1; //which is 0
    diagonalX[1] = 1.0;
    //All last values = 0
    ax[imax] = 0.0;
    bx[imax] = 0.0;
    cx[imax] = 0.0;
    diagonalX[imax] = 1.0;
    //Diagonals, d has full, a misses last, b misses first
    ax[imax-1] = 0.0;// a misses last
    bx[2] = 0.0; //b misses first

    //LINE GAUSS SEIDEL LOOP
    do{
        laplace->ERROR = 0.0;
        // Loop through iTH row
        for (int j = 2; j<=laplace->jmax-1;j++){
            //Loop through jTH row
            for (int i = 2; i<=laplace->imax-1;i++){
                //BOUNDARY CONDITIONS
                //at I = 2, zero
                if (i == 2){
                    cx[i] = -(laplace->OneMinusOmega*laplace-
>TimesBetaSquare*laplace->psi[j][i])
                        -((laplace->OmegaBetaSquare)*(((laplace->psi[j+1][i]))+
                        (laplace->psi_updated[j-1][i])))-
                        ((laplace->w)*(laplace->psi1))
                    ;
                }

                //at J = JMAX -1, will have psi[JMAX], which needs dPsi/dx = 0
                else if (i == laplace->imax-1){
                    cx[i] = -(laplace->OneMinusOmega*laplace-
>TimesBetaSquare*laplace->psi[j][i])
                        -((laplace->OmegaBetaSquare)*(((laplace->psi[j+1][i]))+
                        (laplace->psi_updated[j-1][i])))-
                        ((laplace->w)*(laplace->psi_updated[j][i+1]));
                }

                else{
                    cx[i] =
                        -(laplace->OneMinusOmega*laplace-
>TimesBetaSquare*laplace->psi[j][i])
                        -((laplace->OmegaBetaSquare)*(((laplace->psi[j+1][i]))+
                        (laplace->psi_updated[j-1][i])));
                }
            }
            thomasTriDiagonalX(j,ax,bx,cx,diagonalX,laplace);

            //BC for dPsi/dx = 0;
            laplace->psi_updated[j][laplace->imax] =
                laplace->psi_updated[j][laplace->imax-1];

            for(int i=1; i<=laplace->imax; i++) {
                laplace->ERROR +=
                    abs((laplace->psi_updated[j][i] - laplace->psi[j][i]));
            }
        }

        //Updating Pupdated with P
        for (int j = 1; j<=laplace->jmax;j++){
            for (int i =1; i<=laplace->imax;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }

        //Update the iteration counter
        laplace->iter = laplace->iter + 1;

    }while(laplace->ERROR > laplace->ERROR_MAX);

    t = clock()-t;

    laplace->timeElapsed = (float)t/(CLOCKS_PER_SEC);

    //End clock
    printf("Line SOR Converged! Max iter is: %d, w = %f at %f seconds \n",
laplace->iter,laplace->w, laplace->timeElapsed);
    FILE *outfile3;
    outfile3 = fopen("ResultsLSOR.dat","w");
    for (int j = 1; j<=laplace->jmax;j++){
        for (int i = 1; i<=laplace->imax;i++){
            fprintf(outfile3,"%6.9f\t", laplace->psi[j][i]);
        }
    fprintf(outfile3,"\n");
    }
}


//POINT SOR
void PointSOR(struct Elliptic *laplace){
    //start clock
    clock_t t;
    t = clock();
    do{
        laplace->ERROR = 0.0;
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i = 2; i<=laplace->imax-1;i++){
                    //Finite Difference, using psi_updated as Latest Data

                    laplace->psi_updated[j][i] = ((1-laplace->w)*(laplace->psi
[j][i]))+((laplace->Omegaby2BetaSquare)*(laplace->psi[j][i+1]+laplace-
>psi_updated[j][i-1]+(laplace->betaSquare)*(laplace->psi[j+1][i]+laplace-
>psi_updated[j-1][i])));

                     //Calculate error, keep doing this until satisfy ERROR MAX
                    laplace->ERROR += abs((laplace->psi_updated[j][i] -
laplace->psi[j][i]));
            }
        }

        //Updating Pupdated with P
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i =2; i<=laplace->imax-1;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }
        // Make sure BC satisfied, dpsi/dx = 0
        for (int j = 2; j<=laplace->jmax-1;j++){
                laplace->psi[j][laplace->imax] = laplace->psi[j][laplace-
>imax-1];
        }

        //Update the iteration counter
        laplace->iter = laplace->iter + 1;

        }while(laplace->ERROR >= laplace->ERROR_MAX);

        t = clock()-t;

        laplace->timeElapsed = (float)t/(CLOCKS_PER_SEC);

        printf("POINT SOR Converged! Max iter is: %d, w = %f at %f seconds
\n", laplace->iter,laplace->w, laplace->timeElapsed);

        FILE *outfile4;

        outfile4 = fopen("ResultsPSOR.dat","w");
        for (int j = 1; j<=laplace->jmax;j++){
            for (int i = 1; i<=laplace->imax;i++){
                fprintf(outfile4,"%6.9f\t", laplace->psi[j][i]);
            }
            fprintf(outfile4,"\n");
        }


}


void PrintTablesAndCompare(){
    // TABLE FOR BLOWING UP
    //FOR POINT SOR
    {   int nw = 20;
        double dw = 0.1;
        double w0 = 0.1;
        float wi;
        FILE *testPSOR;
        testPSOR = fopen("testPSOR.dat","w");
        FILE *neatTablePSOR;
        neatTablePSOR = fopen("neatTablePSOR.txt","w");
        fprintf(neatTablePSOR,"  w     |  Iteration  | Time[sec] |\n");
        fprintf(neatTablePSOR,"----------------------------------\n");
        for(int i=0; i<nw; i++){
            Elliptic PSOR;
            wi = w0 + dw*( (float) i);
            InitializePsi(&PSOR,wi);
            PointSOR(&PSOR);
            fprintf(neatTablePSOR, "%.3f\t|\t%4.0d\t  |\t%f  |
\n",PSOR.w,PSOR.iter,PSOR.timeElapsed);
            fprintf(testPSOR, "%.3f\t%4.0d\t%f
\n",PSOR.w,PSOR.iter,PSOR.timeElapsed);
        }
        fprintf(neatTablePSOR,"\nTable 1- Different values of relaxation
parameter (w) for Point SOR\n\n");
        fclose(testPSOR);
        fclose(neatTablePSOR);
    }
    printf("\n");
    //FOR LINE SOR
    {   int nw = 20;
        double dw = 0.1;
        double w0 = 0.1;
        float wi;
        FILE *testLSOR;
        testLSOR = fopen("testLSOR.dat","w");
        FILE *neatTableLSOR;
        neatTableLSOR = fopen("neatTableLSOR.txt","w");
        fprintf(neatTableLSOR,"  w     |  Iteration  | Time[sec] |\n");
        fprintf(neatTableLSOR,"-----------------------------------\n");
        for(int i=0; i<nw; i++) {
            Elliptic LSOR;
            wi = w0 + dw*( (float) i);
            InitializePsi(&LSOR,wi);
            LineSOR(&LSOR);
            fprintf(neatTableLSOR, "%.3f\t|\t%4.0d\t  |\t%f  |
\n",LSOR.w,LSOR.iter,LSOR.timeElapsed);
            fprintf(testLSOR, "%.3f\t%4.0d\t%f
\n",LSOR.w,LSOR.iter,LSOR.timeElapsed);
        }
        fprintf(neatTableLSOR,"\nTable 2- Different values of relaxation
parameter (w) for Line SOR\n\n");
        fclose(testLSOR);
        fclose(neatTableLSOR);
    }
}

//END HEADER FILE WITH FUNCTIONS
```

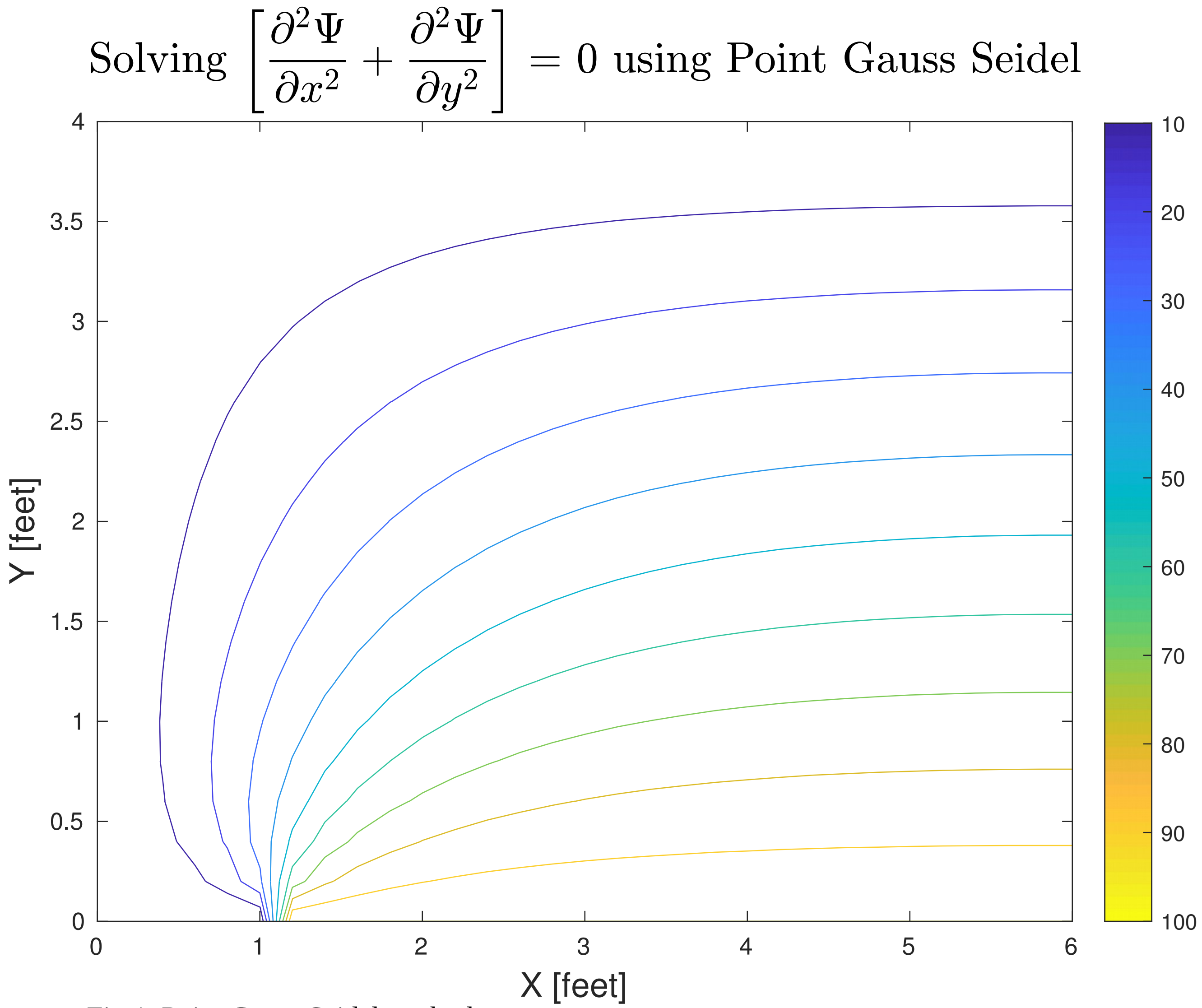


Fig 1. Point Gauss Seidel method

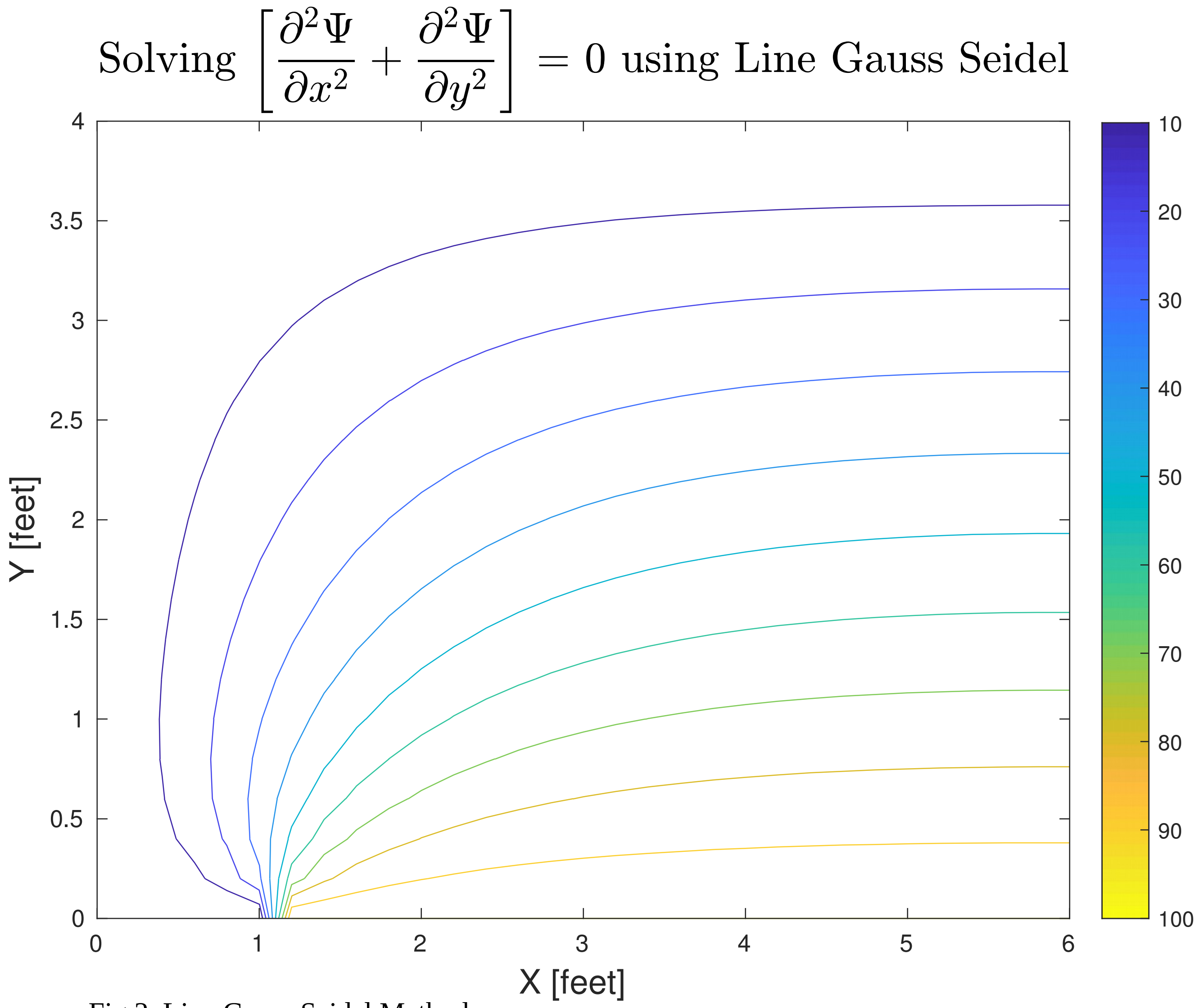


Fig 2. Line Gauss Seidel Method

```
  w     |  Iteration  | Time[sec] |
-----------------------------------
0.100   |   9813      | 0.289863  |
0.200   |   5141      | 0.140544  |
0.300   |   3430      | 0.096141  |
0.400   |   2524      | 0.070281  |
0.500   |   1958      | 0.053929  |
0.600   |   1567      | 0.042769  |
0.700   |   1279      | 0.034924  |
0.800   |   1058      | 0.028872  |
0.900   |    881      | 0.024065  |
1.000   |    737      | 0.020121  |
1.100   |    616      | 0.016729  |
1.200   |    512      | 0.013936  |
1.300   |    423      | 0.011516  |
1.400   |    344      | 0.009367  |
1.500   |    274      | 0.007498  |
1.600   |    210      | 0.005722  |
1.700   |    151      | 0.004122  |
1.800   |     92      | 0.002550  |
1.900   |     99      | 0.002697  |
2.000   |   39686     | 1.085977  |
```

Table 1- Different values of relaxation parameter (w) for Point SOR

It can be seen that at around w = 1.8 and w = 1.9, the solution converges faster (less iterations and less computational time). After testing with different values between 1.8 and 1.9, it is decided that **w = 1.801** is the optimum relaxation parameter. In the code, setting w = 1.801 takes 91 iterations and 0.002460 seconds.

```
  w     |  Iteration  | Time[sec] |
-----------------------------------
0.100   |   9668      | 0.397302  |
0.200   |   4931      | 0.188407  |
0.300   |   3191      | 0.119139  |
0.400   |   2267      | 0.089969  |
0.500   |   1685      | 0.063040  |
0.600   |   1283      | 0.048021  |
0.700   |    984      | 0.036855  |
0.800   |    753      | 0.028300  |
0.900   |    567      | 0.021368  |
1.000   |    413      | 0.015536  |
1.100   |    283      | 0.010716  |
1.200   |    171      | 0.006466  |
1.300   |     83      | 0.003198  |
1.400   |   2564      | 0.095974  |
1.500   |    566      | 0.021152  |
1.600   |    334      | 0.012560  |
1.700   |    232      | 0.009338  |
1.800   |    170      | 0.007004  |
1.900   |    121      | 0.005359  |
2.000   |     61      | 0.002851  |
```

Table 2- Different values of relaxation parameter (w) for Line SOR

A similar observation can be said for the Line SOR case: at **w = 1.3**, the solution converges the fastest (at 83 iterations and 0.003198 seconds). It can also be seen from later plots of relaxation parameter vs. iteration numbers, that anything higher than the optimum value will not guarantee convergence. As a result, w = 1.3 is chosen as the optimal value for Line SOR.

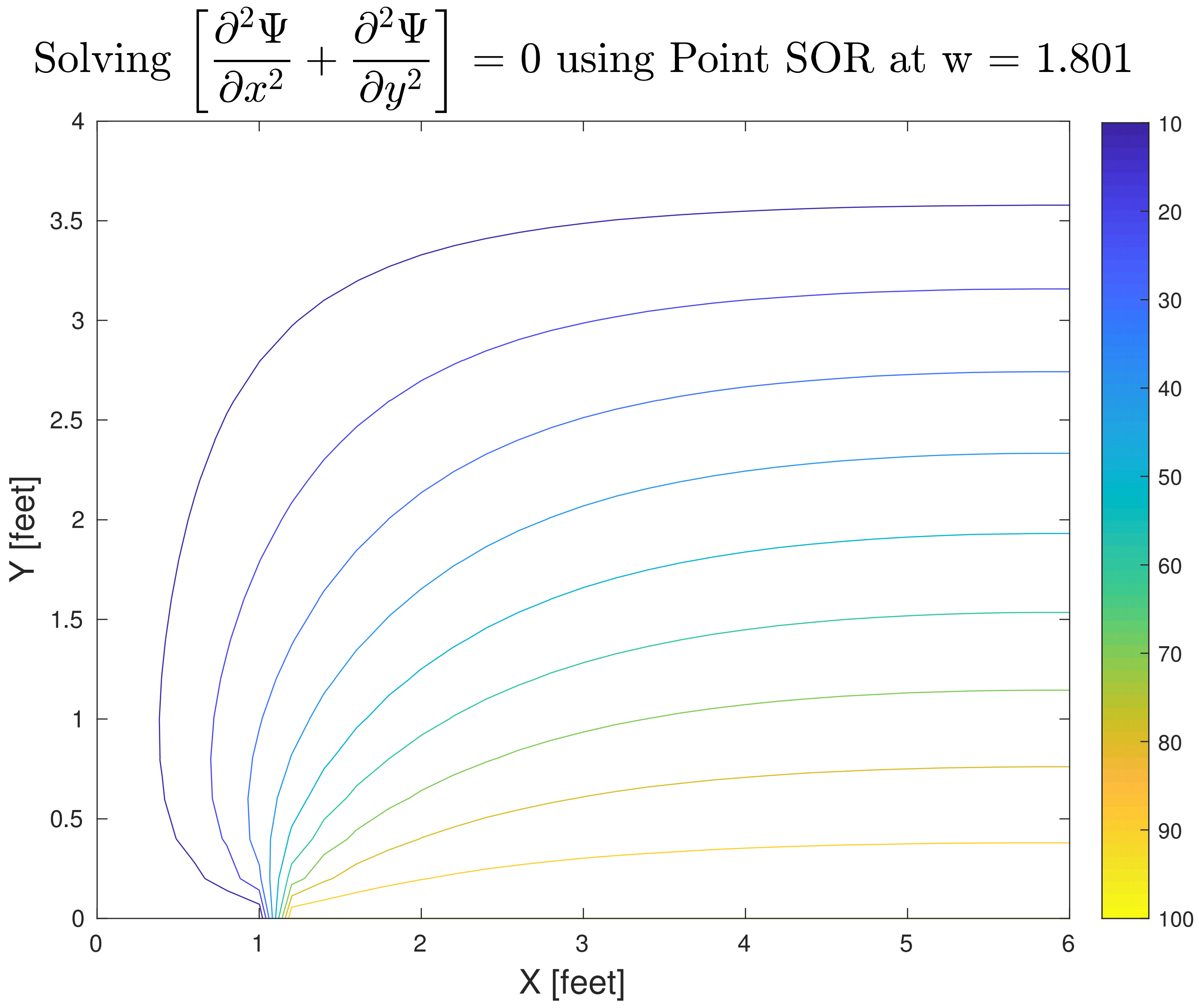


Fig 3. Point SOR Method at optimum relaxation parameter

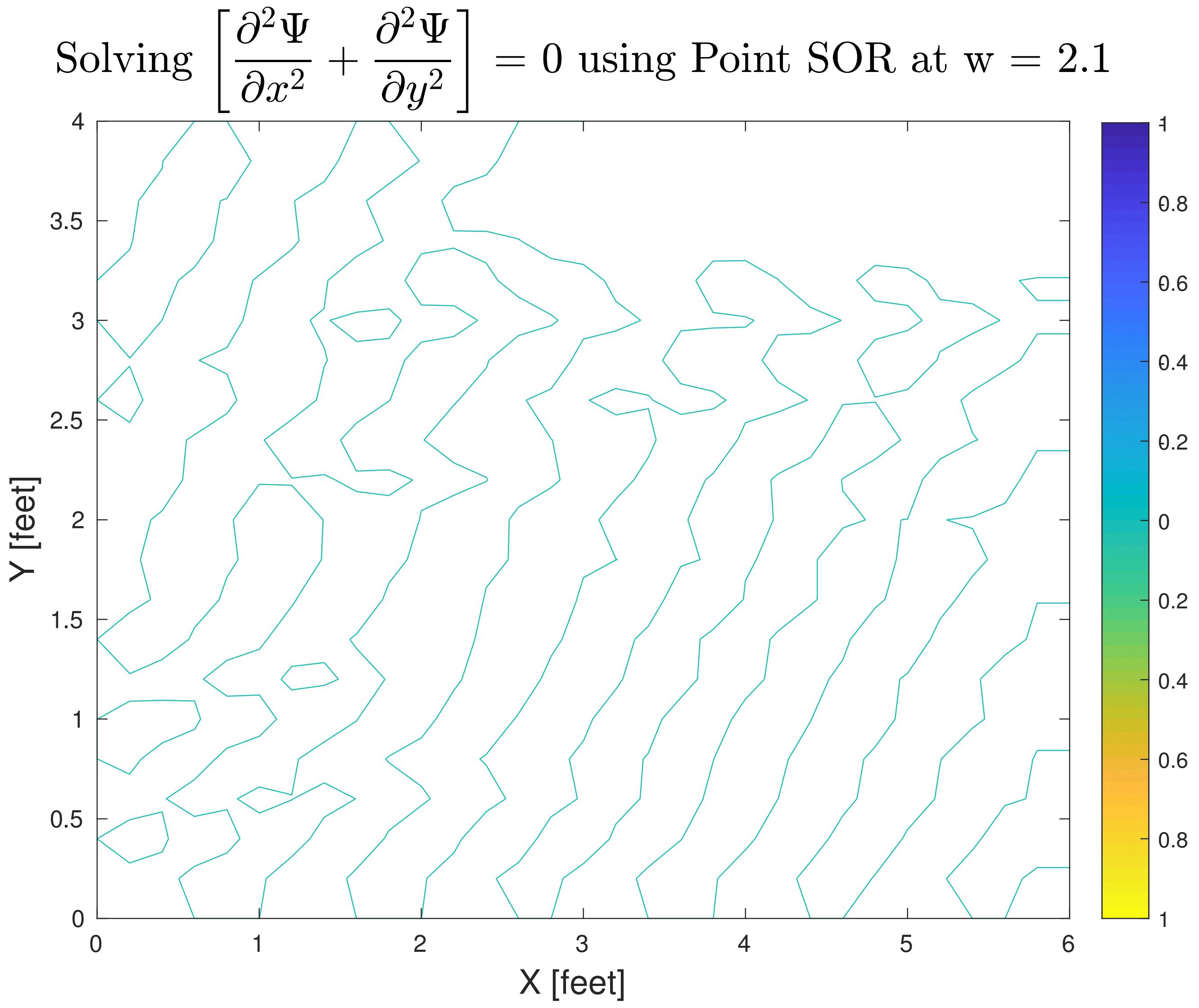


Fig 4. Point SOR Method at very high w.

At w>2, the solution blows up and diverges instead.

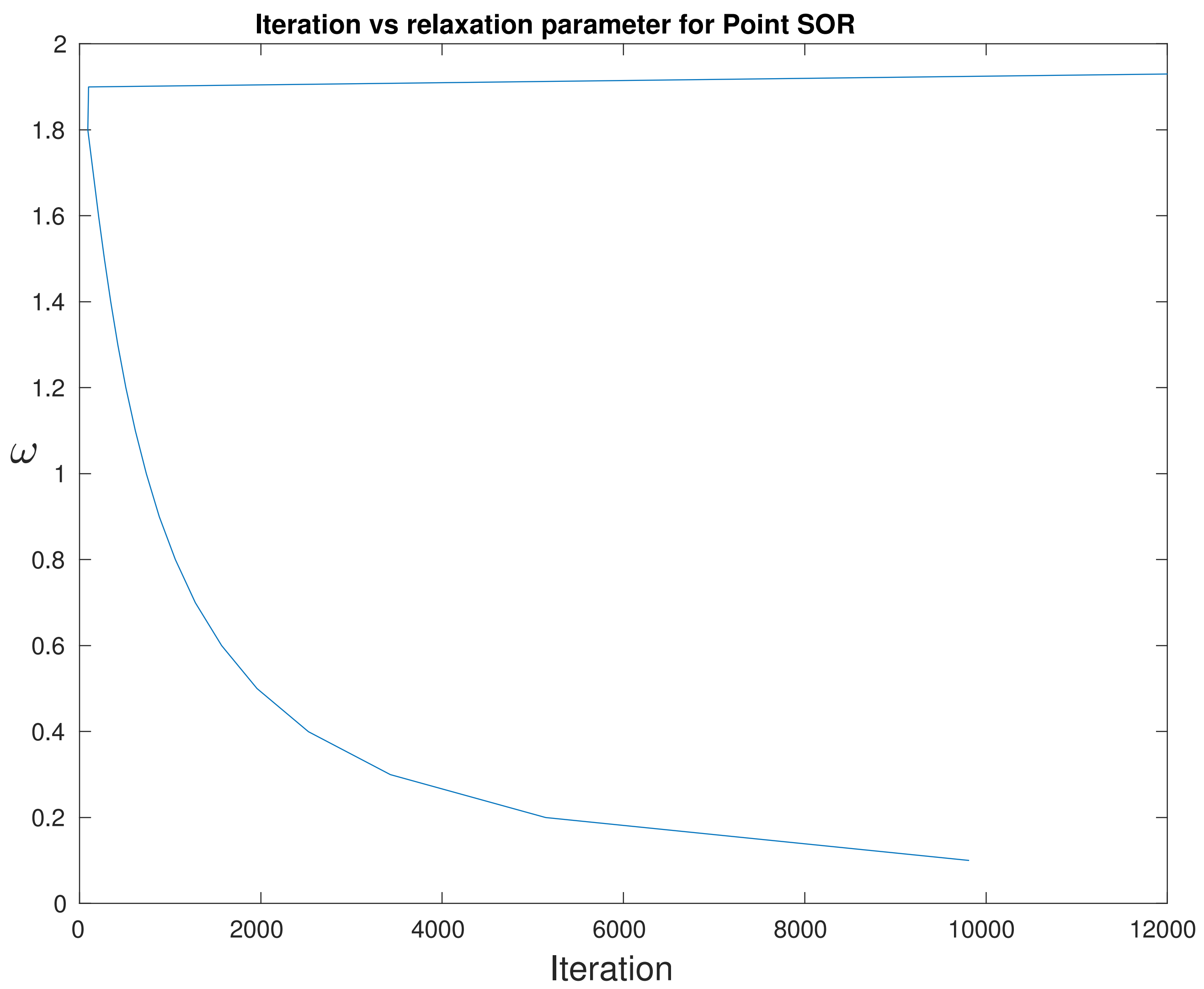


Fig 5. Iteration vs Relaxation Parameter for Point SOR Method

It can be seen that higher W will give lower Iterations. However, very high W will cause the Iterations to rise and from later plots, the solution can also diverge. The optimum relaxation parameter is shown clearly in this plot, where 1.8<w<1.9.

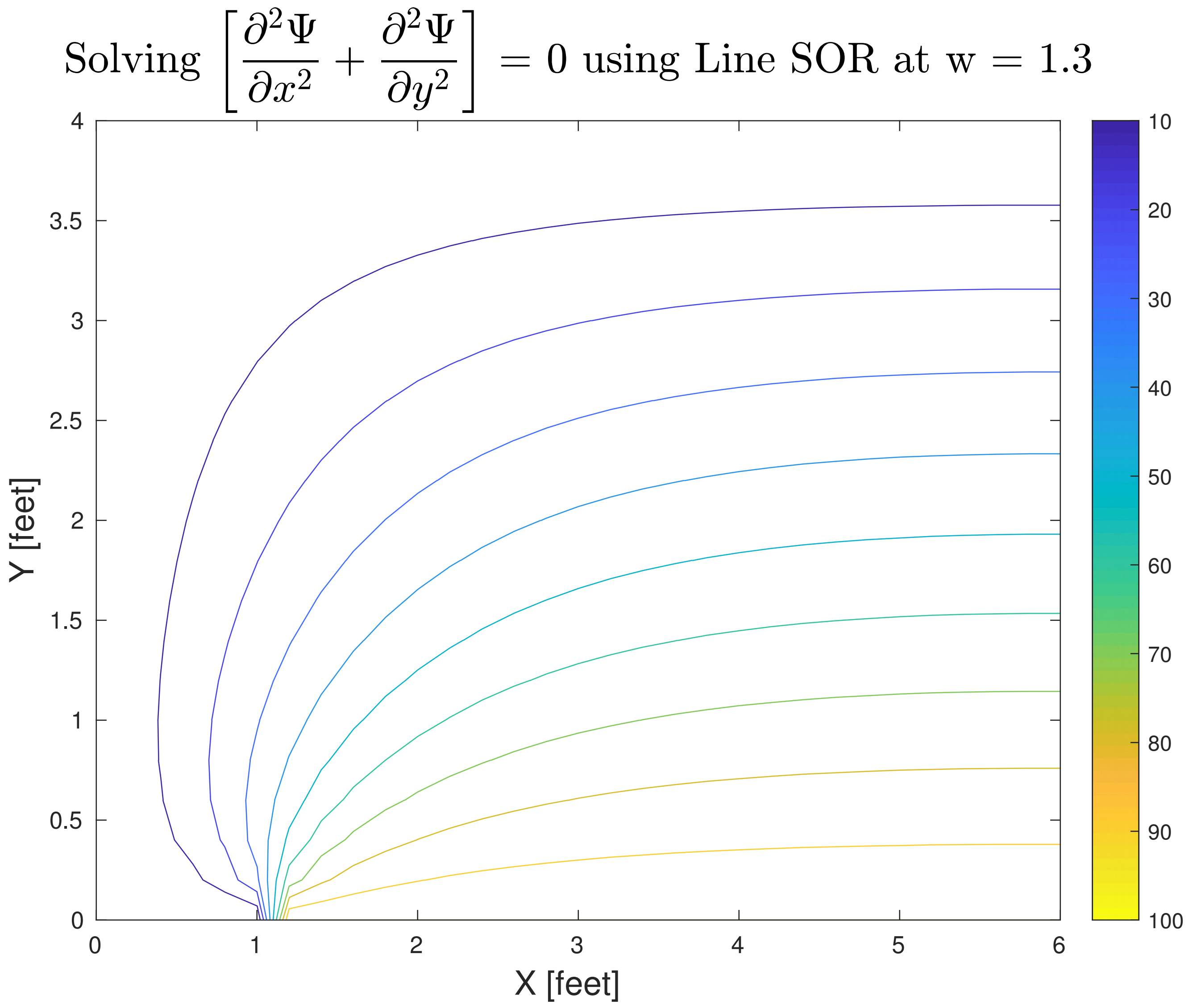


Fig 6. Linet SOR Method at optimum relaxation parameter

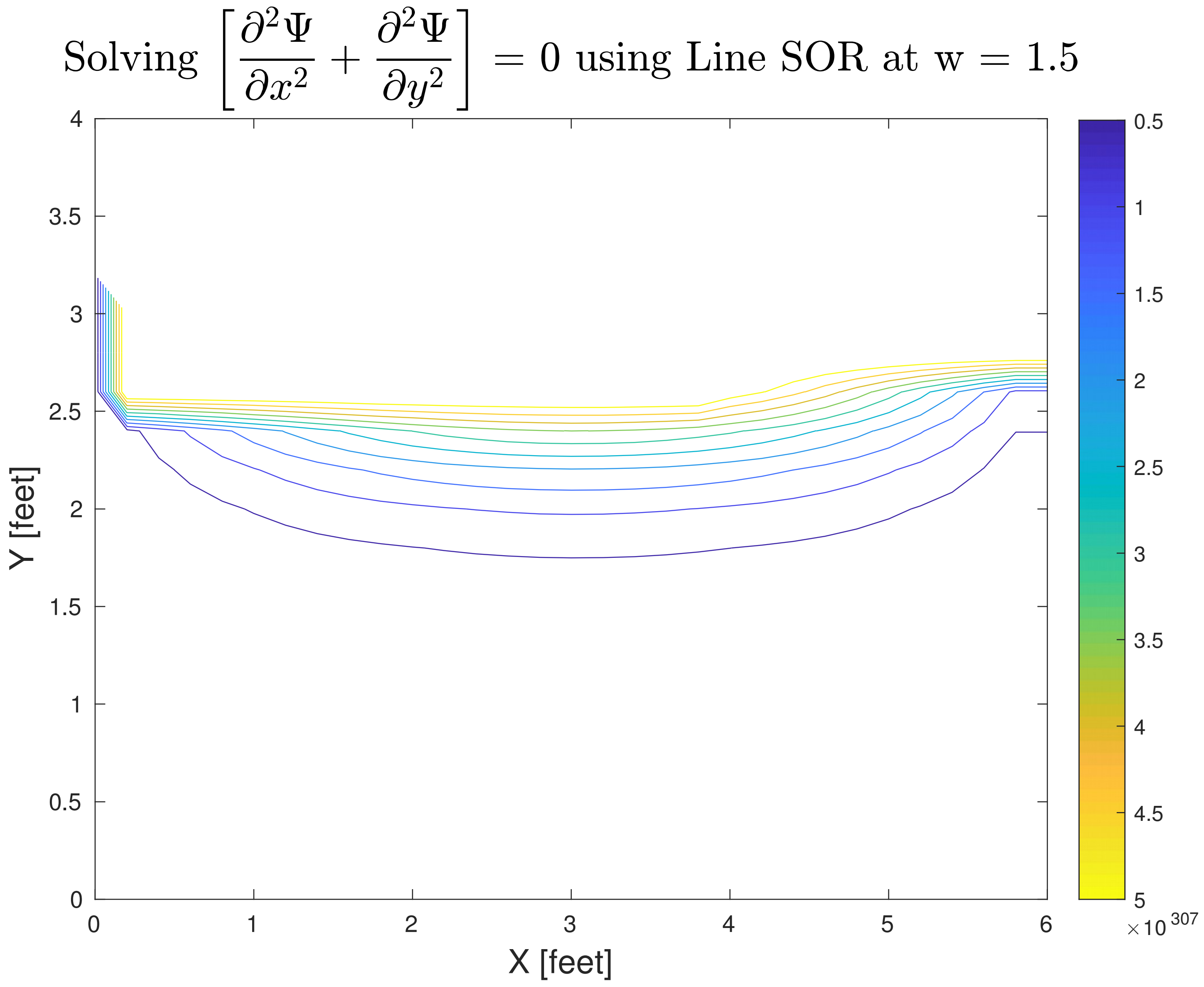


Fig 7. Point SOR Method at very high relaxation parameter

For Line SOR, the solution reaches optimum convergence at a lower w than Point SOR. Still, passing this optimal value causes divergence. We can see that the code tries to keep the derivative boundary condition (dPsi/dx = 0) on the right hand side, but due to the divergence of earlier data points, this does not help it to converge.

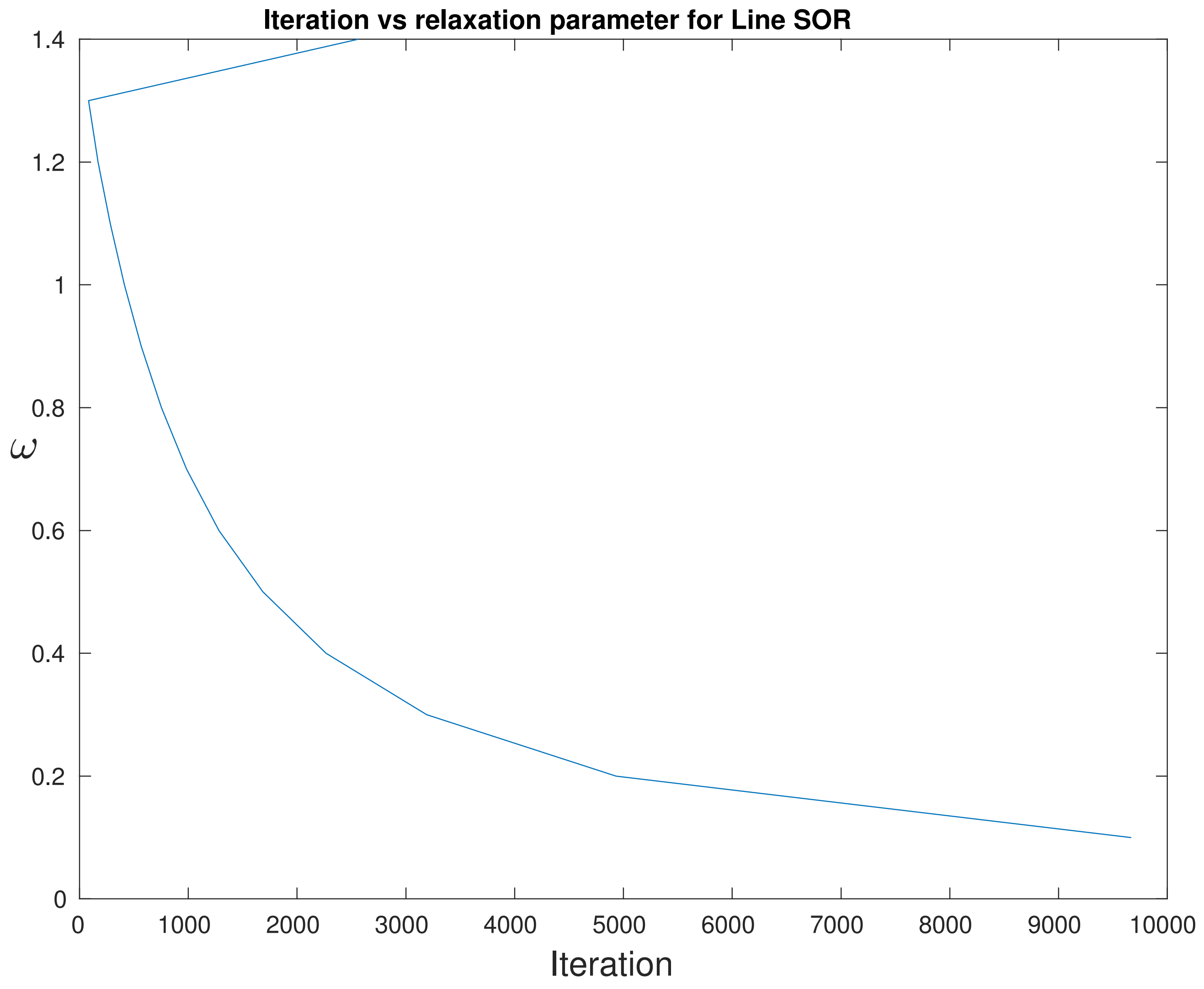


Fig 8. Iteration vs Relaxation Parameter for Line SOR Method

Line SOR converges faster than Point SOR, and also reaches optimal value of relaxation parameter faster. The same trend can be observed: higher W will give lower Iterations, but very high W will cause divergence. It can also be seen clearly that w = 1.3 is the optimum value for this scheme (less iterations).