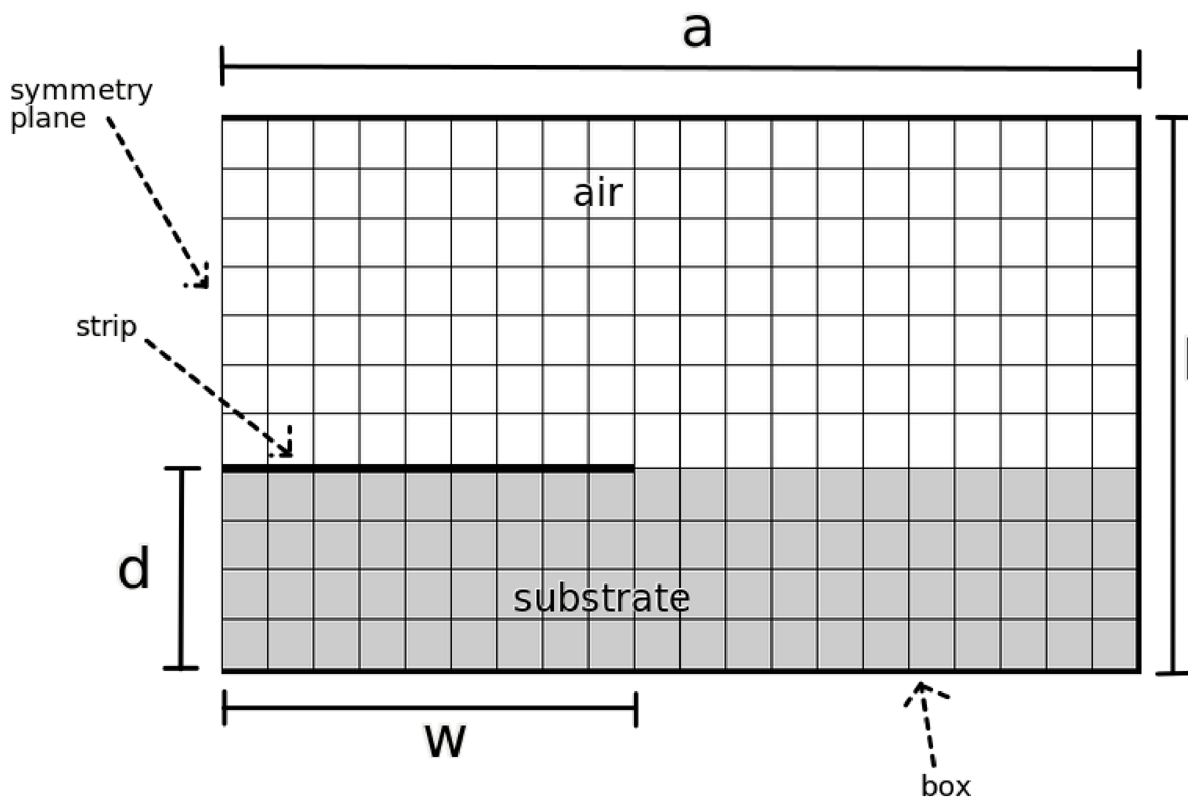
ECEN 445 FDSOR Computer Project

Coady Lewis
04-08-2021

Structure Geometry)



Parameter Description)

“a” is half of the box width; “b” is the height of the box; “d” is the thickness of the substrate; and “w” is half of the strip width. Each vertical step in the grid is represented by “k”, and each horizontal step is represented by “h”. For reference, in the example grid above, d=4k, w=9h, a=20h, and b=11k. Note: h and k are not necessarily equal. They are determined by dividing a and b by n_a and n_b, respectively (where n_a is the number of horizontal steps in a and n_b is the number of vertical steps in b).

FDSOR Method and Equations)

This method is an iterative process that adjusts the potential at each point in the grid until a solution satisfying both Maxwell’s equations and the structure’s boundary conditions is found. Since the method uses finite difference approximations, the solution is only valid at the points in the grid, so any graphing or calculations involving the potential distribution must be discrete processes. For the same reason, the box and strip absolutely must lie in the grid.

The process begins by initializing the grid to 0 V at every point except those on the strip. It is only required that the box be at 0 V, but for simplicity, we will start the interior points at 0 V as well. The points on the strip are initialized to 1 V. Each pass will begin at the bottom left in the substrate, and will proceed from left to right in each row. The points on the box and strip are left untouched to maintain the boundary condition. At each point, the residual is calculated using

$$R_P = \frac{1}{2(1+\alpha^2)}\left(\phi_L + \phi_R + \frac{2\alpha^2}{1+\epsilon_r}\left(\phi_A + \epsilon_r \phi_B\right)\right) - \phi_P^{(old)} \quad \alpha = \frac{h}{k}$$

Where the most up to date potential of the points left, right, above, and below the current position is used. On the left boundary, $\phi_L = \phi_R$ due to symmetry.

(Note: The absolute value of each R_P is stored until the iteration is complete to calculate the maximum)

The current position is updated with $\phi_P^{(new)} = \phi_P^{(old)} + \Omega R_P$

where $1 \leq \Omega < 2$ is a relaxation constant used to accelerate convergence.

The relaxation value with the fastest convergence is approximated by

$\Omega_{OPT} \simeq 2(1 - \frac{\pi}{\sqrt{2}}\sqrt{n_a^{-2} + n_b^{-2}})$ and this is used in the code.

After each iteration, the maximum residual is compared with the tolerance value of 10^{-5}. If the maximum residual is less than the tolerance, the distribution is considered converged. If the maximum residual is more than the tolerance, another iteration starts.

After the potential distribution converges, the per-unit-length capacitance is found by the following method.

$$-\frac{C}{\epsilon_0} \simeq \alpha \sum_{m=1}^{m_r}{}' \left[\epsilon_r\left(\phi_{m,n_b-1} - \phi_{m,n_b+1}\right) + \left(\phi_{m,n_a+1} - \phi_{m,n_a-1}\right)\right]$$

$$+ \frac{1}{\alpha}\left[\sum_{n=n_b}^{n_s}{}' \epsilon_r\left(\phi_{m_r+1,n} - \phi_{m_r-1,n}\right) + \sum_{n=n_s}^{n_a}{}' \left(\phi_{m_r+1,n} - \phi_{m_r-1,n}\right)\right]$$

where the prime means the first and last term of each sum is halved.

$$\phi_{m,n} = \phi(x_m, y_n)$$

where $x_1, x_2, ..., x_{m_r}$ and $y_{n_b}, ..., y_{n_s}, ..., y_{n_a}$ denote the points used in the contour. In the case of this code, the closest contour to the strip is used (steps=1), but the contour's distance from the strip can be modified by changing the "steps" parameter in the code.

Once the capacitance is found, the effective dielectric constant (for substrate lines) and the characteristic impedance are found with

$Z_0 = \frac{120\pi}{\sqrt{\frac{C}{\epsilon_0}\cdot\frac{C_0}{\epsilon_0}}}$ where C_0 is the capacitance for an air-line with equivalent geometry.

$$\epsilon_{EFF} = \frac{C}{C_0}$$

Numerical Results)

Line 1 (symmetric air-line) ($w/d=3$, $b/d=2$, $a/w=3$, $\varepsilon_R=1$) ($C_0=C$ in the Z_0 calculation)

N	Iterations	Normalized C	Ω_{OPT}	Z_0 (Ω)
60	99	13.9231	1.8953	27.0767
120	206	13.8430	1.9476	27.2333
240	421	13.8038	1.9738	27.3107
480	848	13.7845	1.9869	27.3489

Exact Value)

$$\frac{C}{\epsilon_0} = 8\left(\frac{w}{b} - \frac{1}{\pi} ln\left(1 + coth\left(\pi\left(\frac{a}{b} - \frac{w}{b}\right)\right)\right)\right) \qquad (w/b)=(w/d)/(b/d)=1.5$$

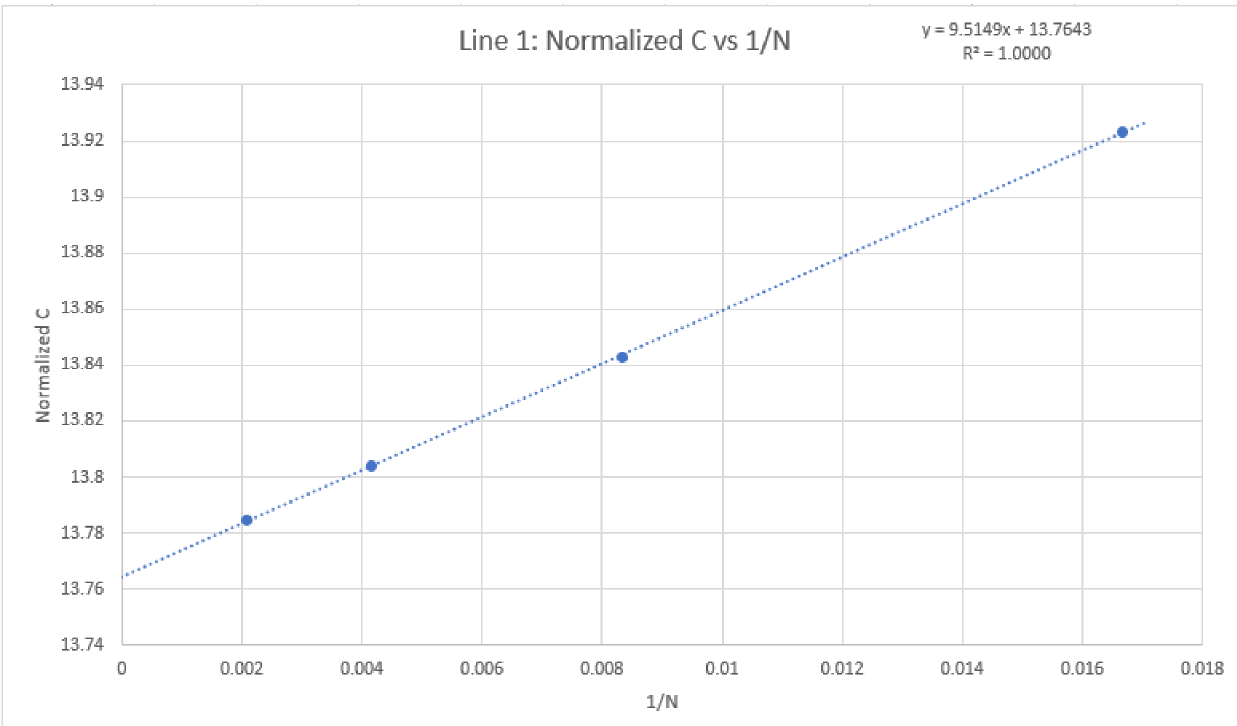
$(a/b)=(a/w)(w/b)=4.5$ $\qquad \frac{C}{\epsilon_0} = 8\left(1.5 - \frac{1}{\pi} ln(1 + coth(3\pi))\right) \approx 13.7651$

Line 2 (air-line) ($w/d=3$, $b/d=10$, $a/w=3$, $\varepsilon_R=1$) ($C_0=C$ in the Z_0 calculation)

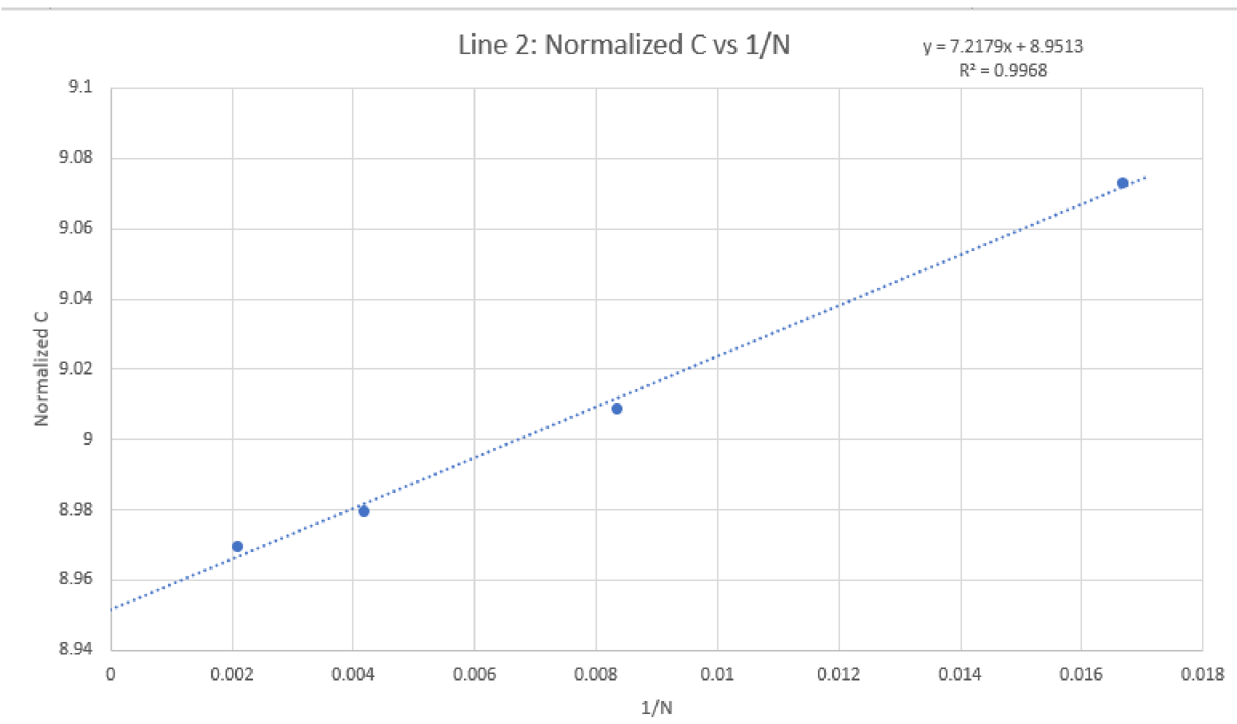
N	Iterations	Normalized C	Ω_{OPT}	Z_0 (Ω)
60	170	9.0730	1.8953	41.5509
120	292	9.0089	1.9476	41.8465
240	506	8.9796	1.9738	41.9831
480	875	8.9694	1.9869	42.0308

Line 3 ($w/d=3$, $b/d=10$, $a/w=3$, $\varepsilon_R=10$) (C_0 is taken from C in Line 2 in the Z_0, ε_{EFF} calculation)

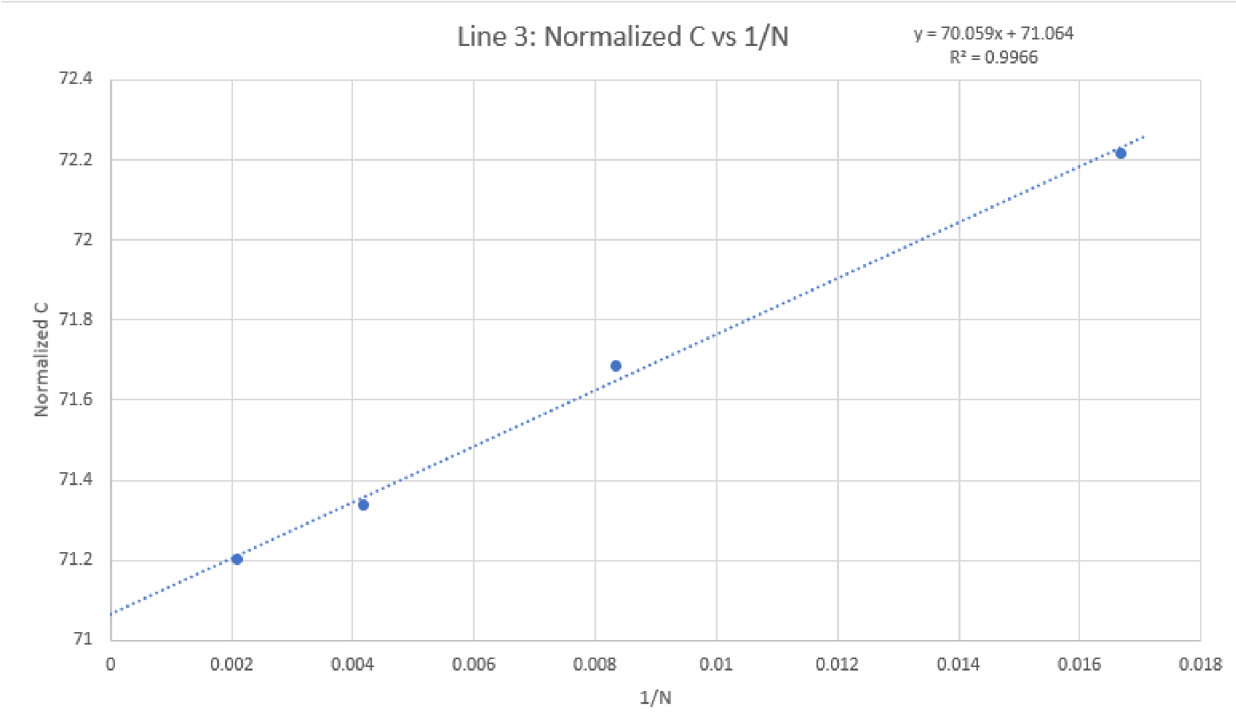
N	Iterations	Normalized C	Ω_{OPT}	ε_{EFF}	Z_0 (Ω)
60	161	72.2169	1.8953	7.9595	14.7277
120	277	71.6874	1.9476	7.9574	14.8345
240	481	71.3403	1.9738	7.9447	14.8948
480	856	71.2007	1.9869	7.9382	14.9179



Line 1: Normalized C vs 1/N
y = 9.5149x + 13.7643
R² = 1.0000
Normalized C
1/N
13.94
13.92
13.9
13.88
13.86
13.84
13.82
13.8
13.78
13.76
13.74
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018



Line 2: Normalized C vs 1/N
y = 7.2179x + 8.9513
R² = 0.9968
Normalized C
1/N
9.1
9.08
9.06
9.04
9.02
9
8.98
8.96
8.94
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018



All 3 lines had a surprisingly linear trend for the (Normalized C) vs (1/N). For all 3 data sets, the linear trendline has an R^2 value greater than 0.99, so the convergence as $N \to \infty$ seems clear, and it is given by the constant term of the trendline equation on each graph.

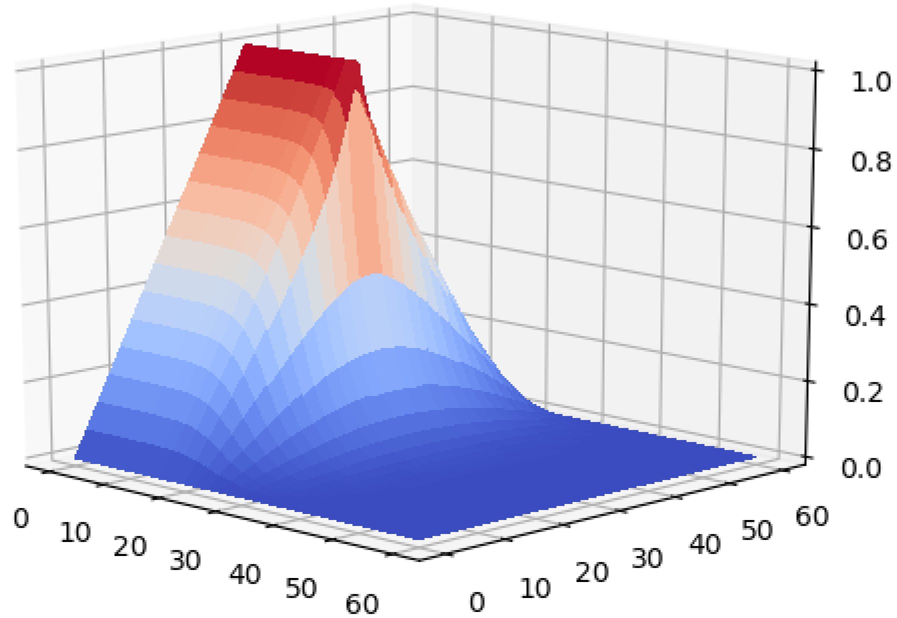
Line 1: Normalized C as $N \to \infty$ = 13.7643
Line 2: Normalized C as $N \to \infty$ = 8.9513
Line 3: Normalized C as $N \to \infty$ = 71.0640

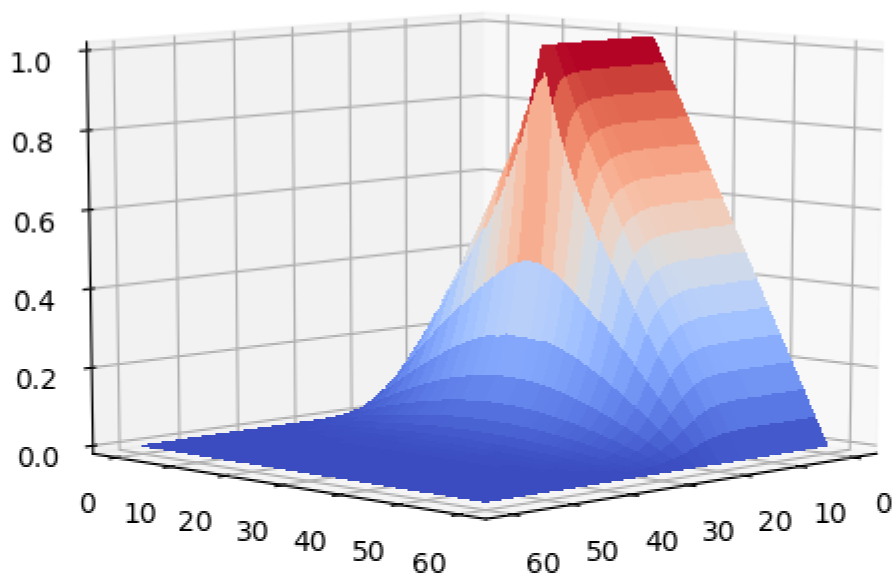
In the case of Line 1, the converged value is only 0.0008 off of the exact value given above.

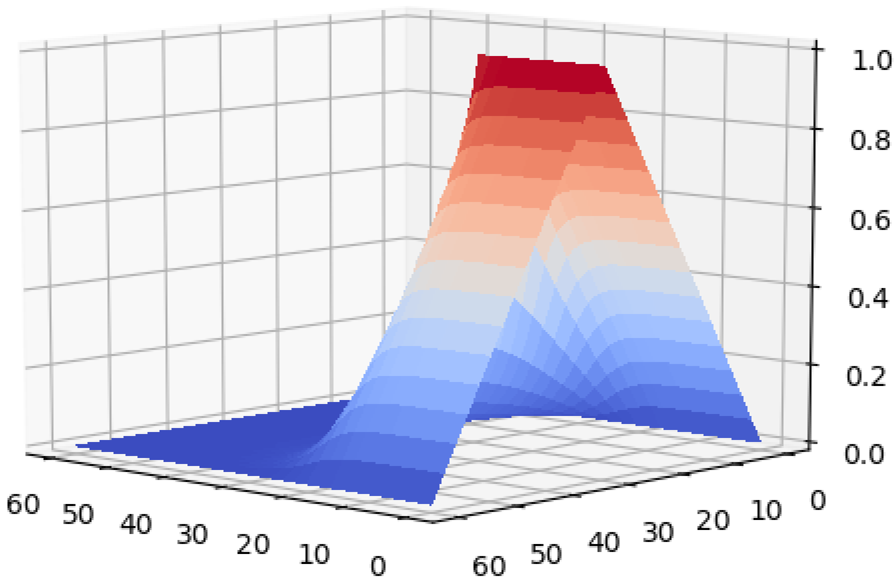
$$\%Error = \frac{|13.7643 - 13.7651|}{13.7651} \cdot 100\% = 5.8118 \cdot 10^{-3}\%$$

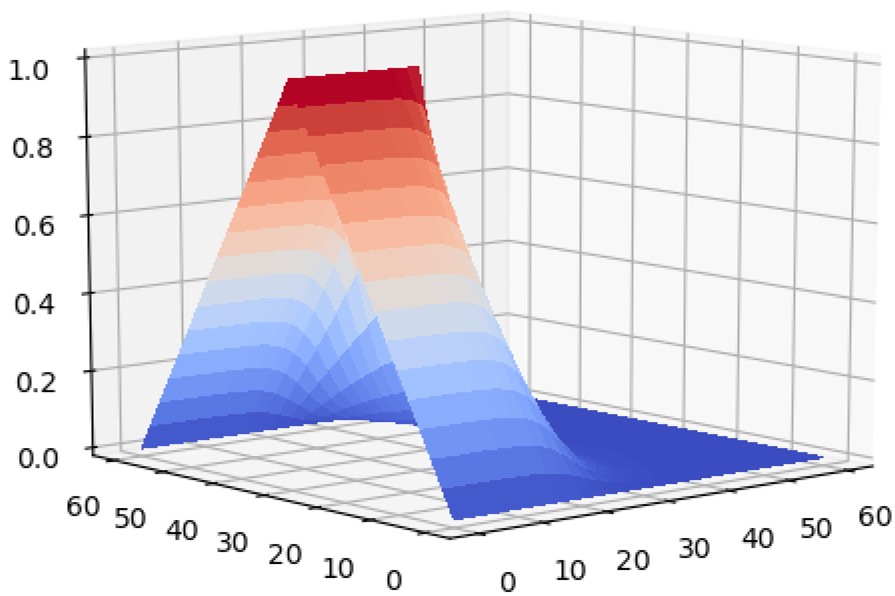
This is extremely close, so this method works very well for large N.

3D Plots with N=60) (View from all 4 corners of the box for each line)
Line 1)

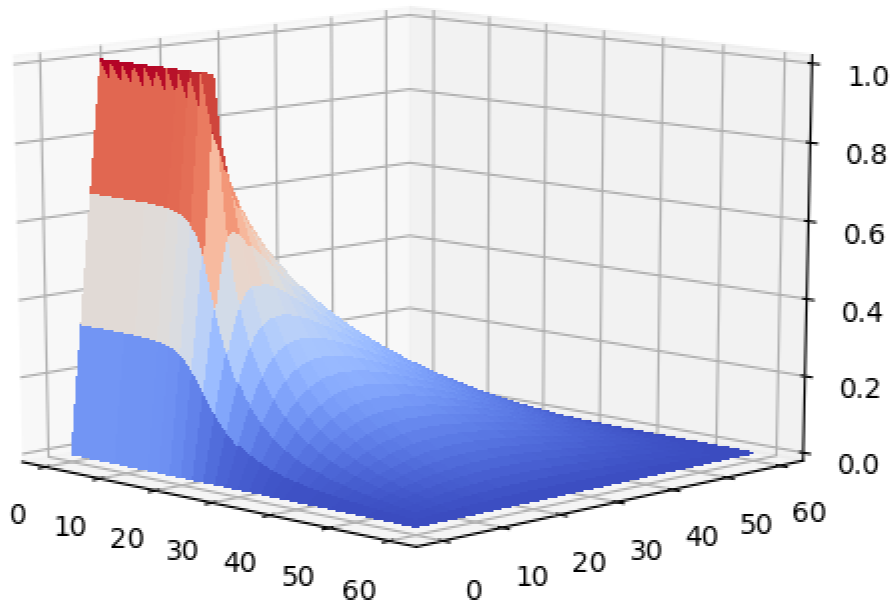


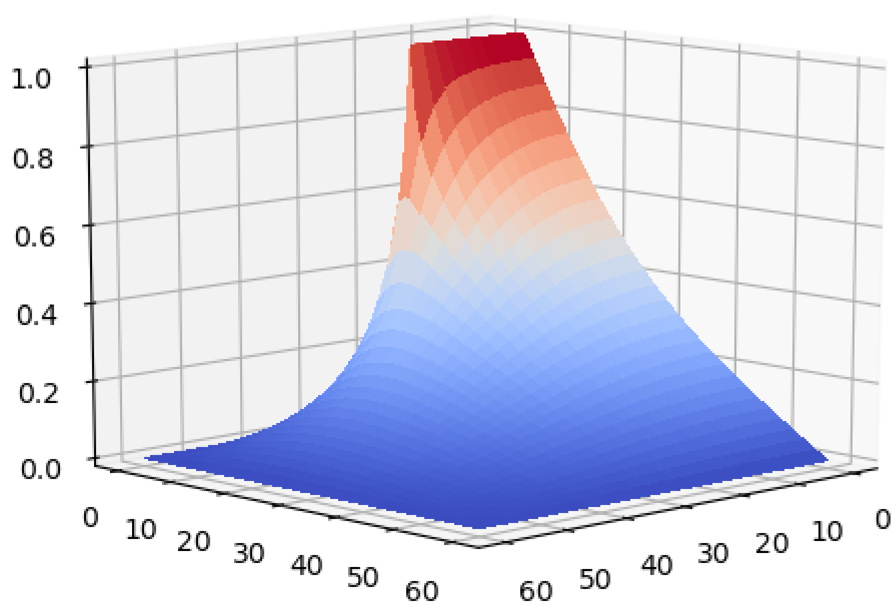


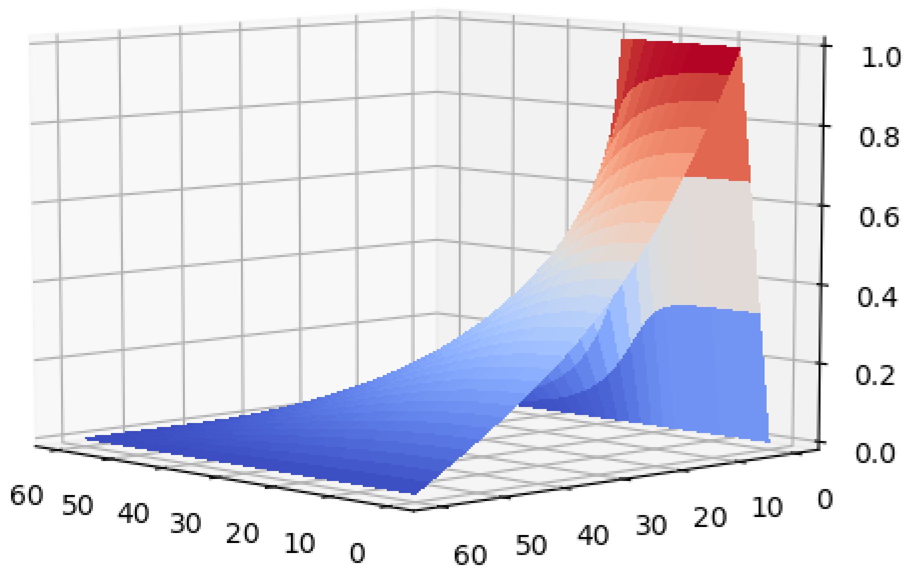

1.0
0.8
0.6
0.4
0.2
0.0
60 50 40 30 20 10 0
60 50 40 30 20 10 0

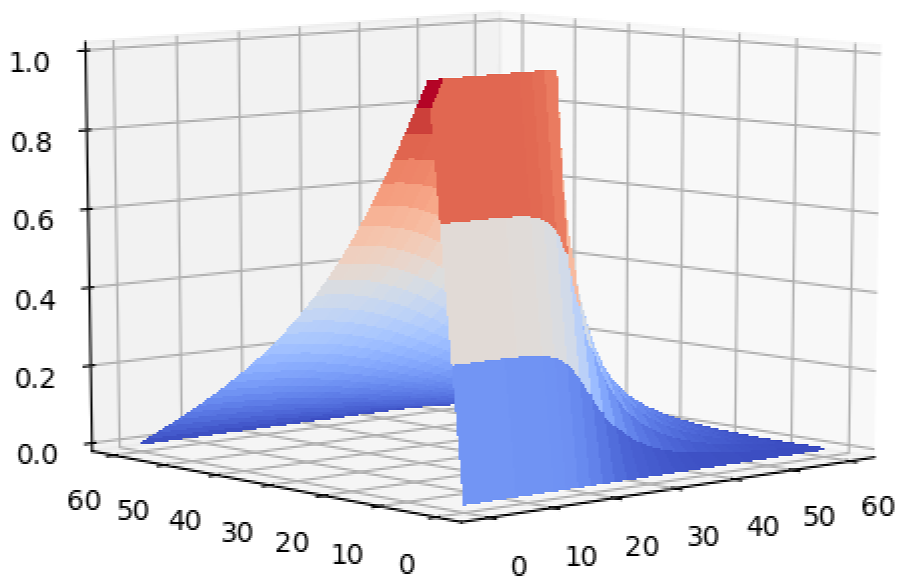

1.0
0.8
0.6
0.4
0.2
0.0
60 50 40 30 20 10 0
0 10 20 30 40 50 60

Line 2)

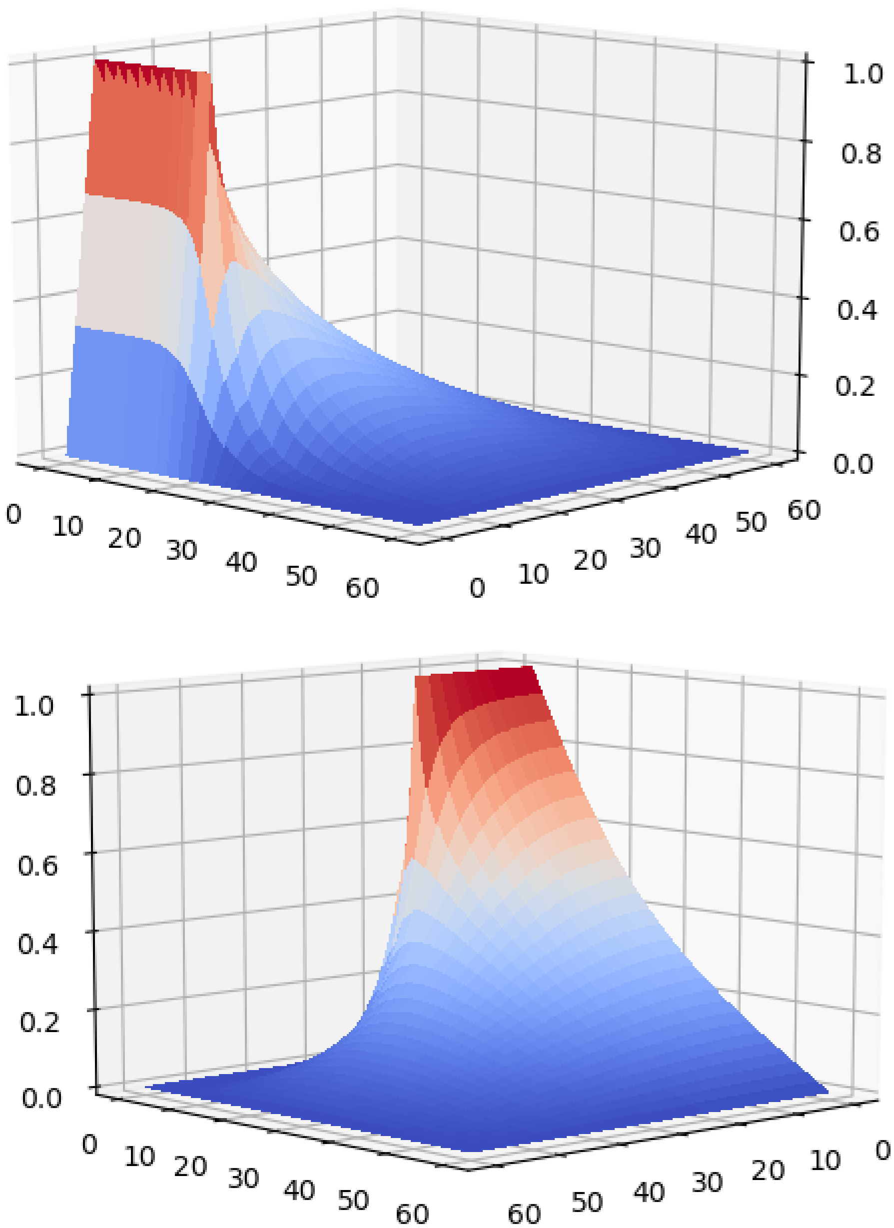

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
0
10
20
30
40
50
60

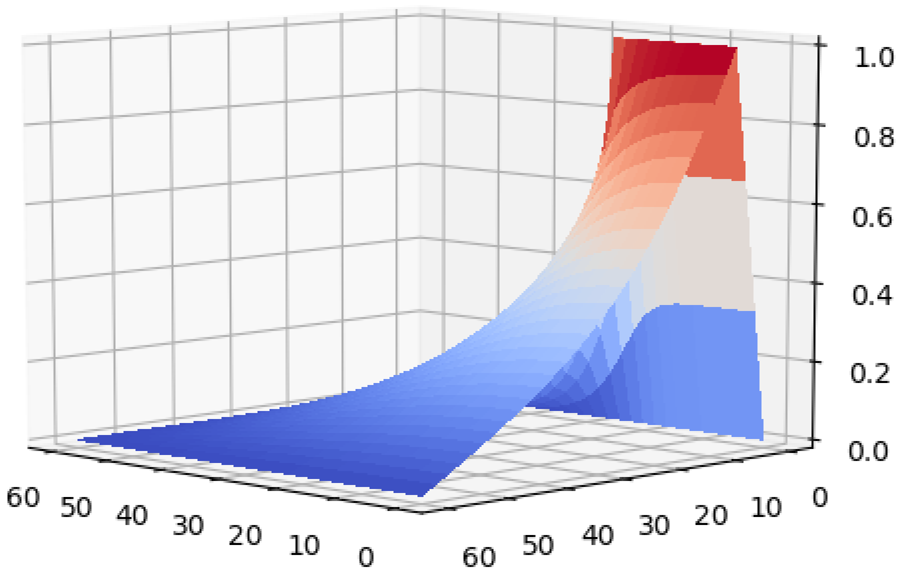

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
60
50
40
30
20
10
0

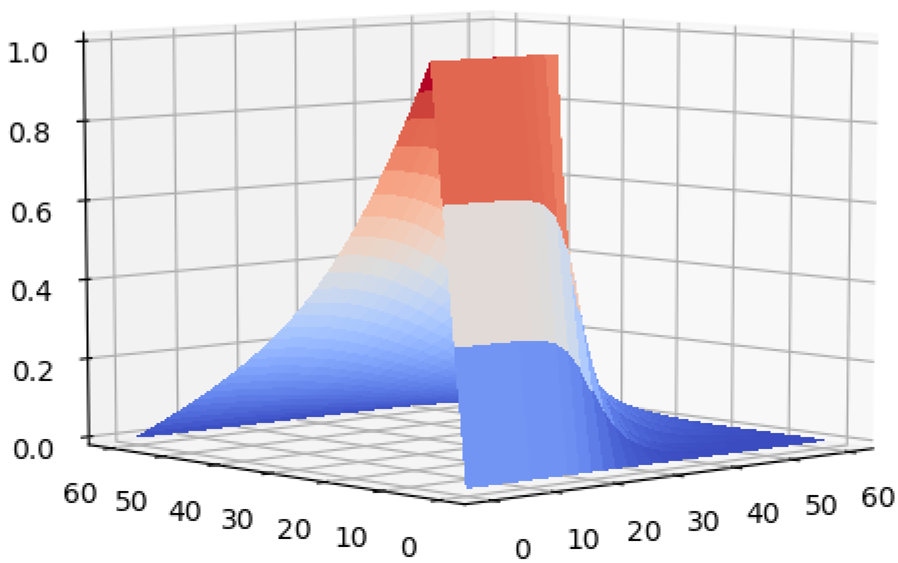

1.0
0.8
0.6
0.4
0.2
0.0
60 50 40 30 20 10 0
60 50 40 30 20 10 0


1.0
0.8
0.6
0.4
0.2
0.0
60 50 40 30 20 10 0
0 10 20 30 40 50 60

Line 3)


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
0
10
20
30
40
50
60
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
60
50
40
30
20
10
0


1.0
0.8
0.6
0.4
0.2
0.0
60 50 40 30 20 10 0
60 50 40 30 20 10 0


1.0
0.8
0.6
0.4
0.2
0.0
60 50 40 30 20 10 0
0 10 20 30 40 50 60

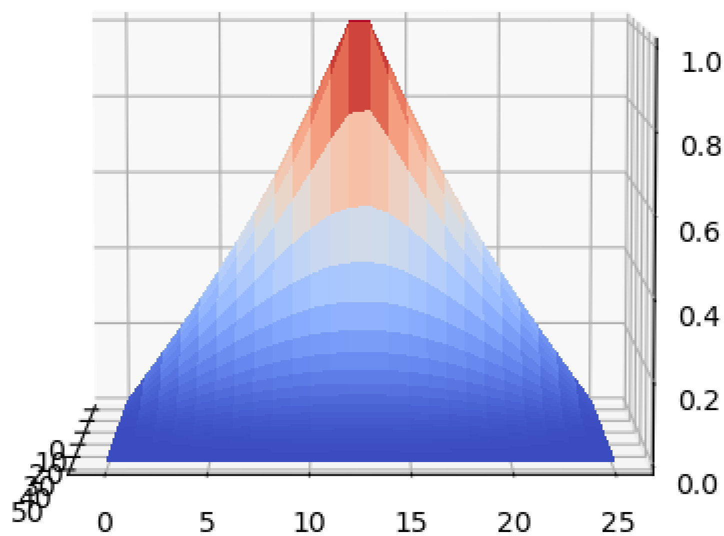
Finite Strip Line Extension) (w/d=5/12, b/d=225/12, a/w=10, ε_R=2.3)

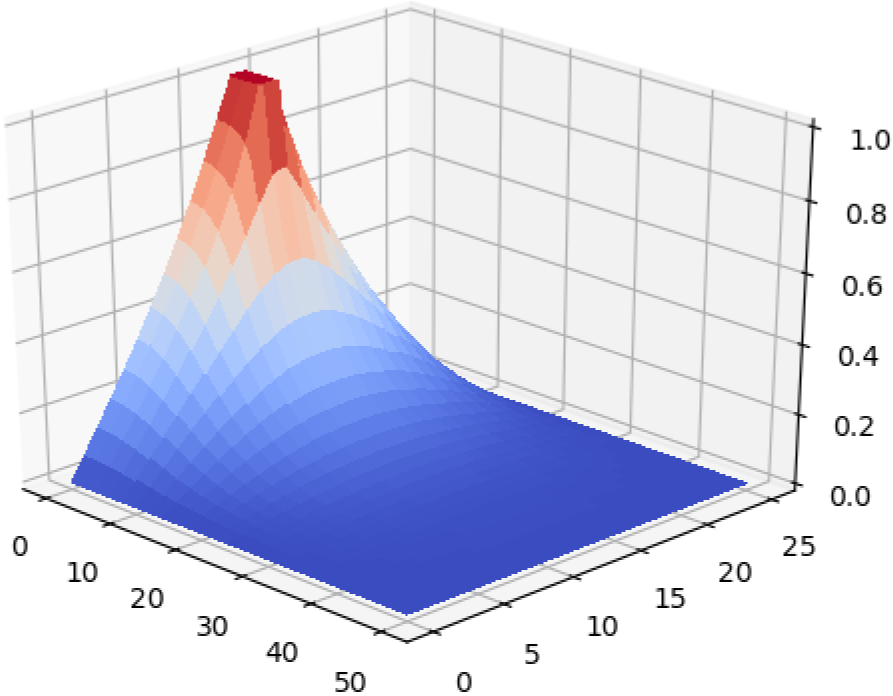
$\Omega_{OPT} = 1.8013$
Normalized C = 5.0932 iterations = 50
Normalized C_0 (set ε_r=1) = 2.8077 iterations = 52

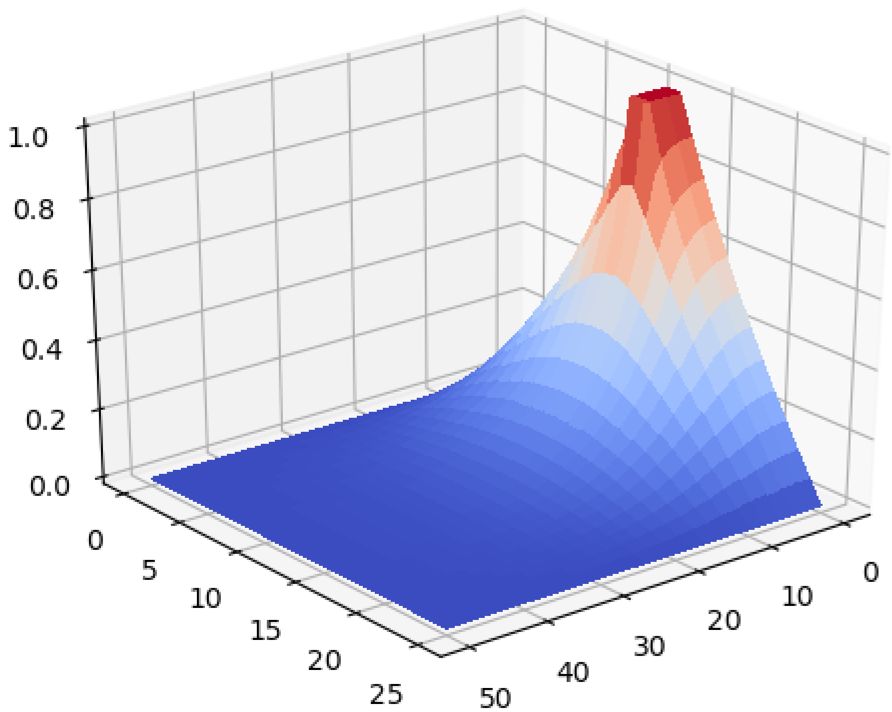
$$Z_0 = \frac{120\pi}{\sqrt{5.0932 \cdot 2.8077}} = 99.6920\ \Omega$$

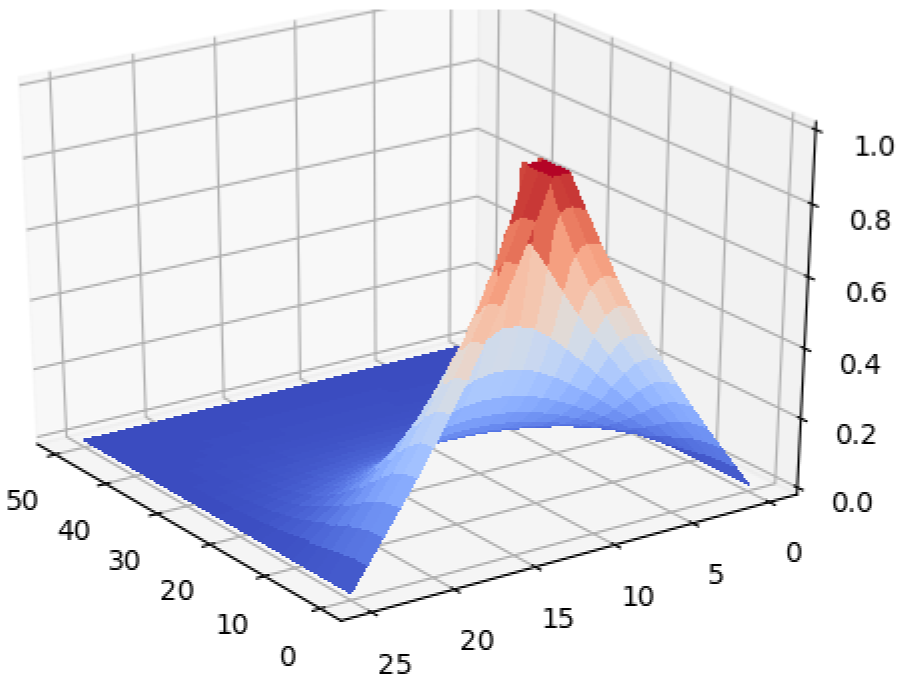
$$\varepsilon_{EFF} = \frac{5.0932}{2.8077} = 1.8140$$

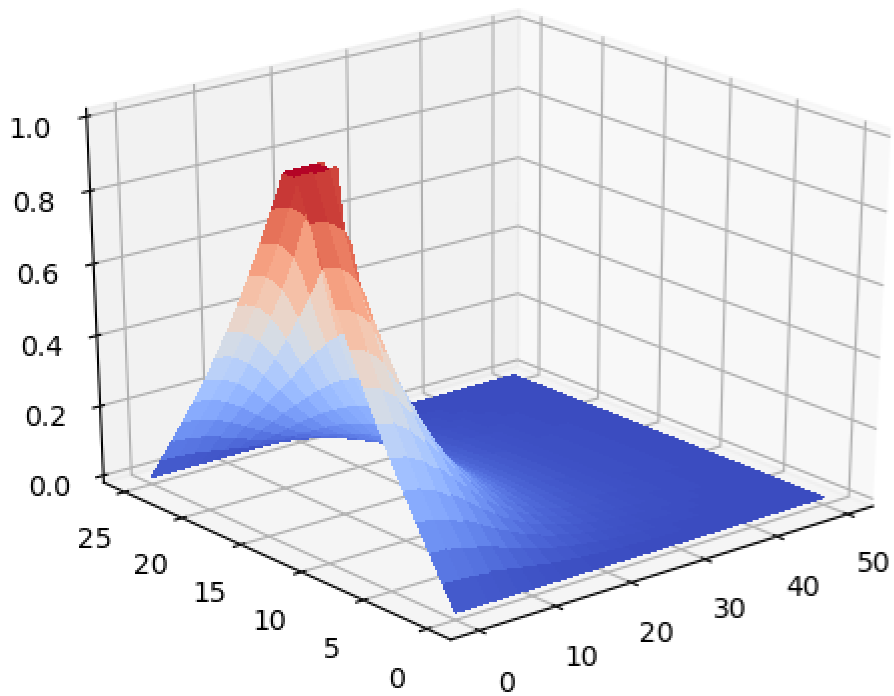
3D Plots) (Strip View, followed by the view from all 4 corners of the box)




0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
0
5
10
15
20
25


0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
0
10
20
30
40
50


1.0
0.8
0.6
0.4
0.2
0.0
50
40
30
20
10
0
25
20
15
10
5
0


1.0
0.8
0.6
0.4
0.2
0.0
25
20
15
10
5
0
0
10
20
30
40
50

Appendix P1)

Code Listing) (.py file is also attached if there are formatting issues)

```
from math import *
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
# Coady Lewis Computer project
# 04-08-2021
# This Code implements the FDSOR method to find
# the potential distribution, and several parameters
# of a stripline


def calc_residual(pot,i,j,al=1,ep=1):
    c1 = 1/(2*(1+al**2))
    c2 = (2*(al**2))/(1+ep)
    if(j==0):
        return c1*(2*pot[i][1]+c2*(pot[i+1][0]+ep*pot[i-1][0]))-pot[i][j]
    else:
        return c1*(pot[i][j-1]+pot[i][j+1]+c2*(pot[i+1][j]+ep*pot[i-1][j]))-pot[i][j]


def rnd(val, d):
    if(val == None):
        return None
    return int(val*(10**d))/(10**d)


def capacitance(pot, sw, sh, al, ep=1, steps=1, finite=False):
    sum1=0
    sum2=0
    sum3=0
    sum4=0
    for i in range(sw+1+steps+int(finite)):
        if(i==0 or i==(sw+steps+int(finite))):
            sum1+=0.5*al*ep*(pot[sh-steps-1][i]-pot[sh-steps+1][i])
            sum2+=0.5*al*(pot[sh+steps+1][i]-pot[sh+steps-1][i])
        else:
            sum1+=al*ep*(pot[sh-steps-1][i]-pot[sh-steps+1][i])
            sum2+=al*(pot[sh+steps+1][i]-pot[sh+steps-1][i])
    for i in range(sh-steps,sh+1):
        if(i==(sh-steps) or i==sh):
            sum3+=0.5*(1/al)*ep*(pot[i][sw+steps+1]-pot[i][sw+steps-1])
        else:
            sum3+=(1/al)*ep*(pot[i][sw+steps+1]-pot[i][sw+steps-1])
    for i in range(sh,sh+steps+1):
        if(i==sh or i==(sh+steps)):
            sum4+=0.5*(1/al)*(pot[i][sw+steps+1]-pot[i][sw+steps-1])
        else:
            sum4+=(1/al)*(pot[i][sw+steps+1]-pot[i][sw+steps-1])
    return (-1)*(sum1+sum2+sum3+sum4)
```

```
def plot(pot):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # Make data.
    X = []
    Y = []
    for i in range(len(pot)):
        X.append(i)
    for i in range(len(pot[0])):
        Y.append(i)
    X,Y = np.meshgrid(Y,X)
    Z=np.array(pot)
    # Plot the surface.
    surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    ax.set_zlim(0, 1)
    plt.show()


def printdist(x,dec=-1):
    #prints matrix with row 0 at the bottom
    for i in range(len(x)-1,-1,-1):
        print(x[i])



def getUserInput():
    # This function gets the problem's parameters from the user
    # and ensures the inputs are of the correct type to proceed
    # without stopping the program.
    out = []
    #
    # Get w/d
    print('\n\nEnter w/d as a ratio of integers')
    while (True):
        try:
            hold = int(input('\n\nEnter the numerator: '))
            while (hold < 1):
                print('\n\nERROR: Input must be positive')
                hold = int(input('\n\nEnter the numerator: '))
            out.append(hold)
            break
        except ValueError:
            print('\n\nERROR: Input must be an integer')
    while (True):
        try:
            hold = int(input('\n\nEnter the denominator: '))
            while (hold < 1):
                print('\n\nERROR: Input must be positive')
                hold = int(input('\n\nEnter the denominator: '))
            out.append(hold)
            break
        except ValueError:
            print('\n\nERROR: Input must be an integer')
```

```
#
# Get b/d and check that b > d
print('\n\nEnter b/d as a ratio of integers')
while (True):
    try:
        hold = int(input('\n\nEnter the numerator: '))
        while (hold < 2):
            # conflict of boundary conditions
            print('\n\nERROR: Input must be at least 2')
            hold = int(input('\n\nEnter the numerator: '))
        out.append(hold)
        break
    except ValueError:
        print('\n\nERROR: Input must be an integer')
while (True):
    try:
        hold = int(input('\n\nEnter the denominator: '))
        while (hold < 1 or hold >= out[2]):
            print('\n\nERROR: Input must be positive and less than the numerator')
            hold = int(input('\n\nEnter the denominator: '))
        out.append(hold)
        break
    except ValueError:
        print('\n\nERROR: Input must be an integer')
#
# Get a/w and check that a > w
print('\n\nEnter a/w as a ratio of integers')
while (True):
    try:
        hold = int(input('\n\nEnter the numerator: '))
        while (hold < 2):
            # conflict of boundary conditions
            print('\n\nERROR: Input must be at least 2')
            hold = int(input('\n\nEnter the numerator: '))
        out.append(hold)
        break
    except ValueError:
        print('\n\nERROR: Input must be an integer')
while (True):
    try:
        hold = int(input('\n\nEnter the denominator: '))
        while (hold < 1 or hold >= out[4]):
            print('\n\nERROR: Input must be positive and less than the numerator')
            hold = int(input('\n\nEnter the denominator: '))
        out.append(hold)
        break
    except ValueError:
        print('\n\nERROR: Input must be an integer')
#
# Get n_a
while (True):
# To position the strip in the mesh, (w/a)*n_a must be an integer
# check that (w*n_a)%a == 0
    try:
```

```
            hold = int(input('\n\nEnter n_a: '))
            while ((hold * out[5]) % out[4] != 0):
                print('\n\nERROR: Entered n_a does not place the end of the strip at a mesh point')
                hold = int(input('\n\nEnter n_a: '))
            while (hold < 1):
                print('\n\nERROR: n_a must be positive')
                hold = int(input('\n\nEnter n_a: '))
            out.append(hold)
            break
        except ValueError:
            print('\n\nERROR: n_a must be an integer')
    #
    # Get n_b
    while (True):
    # To position the strip in the mesh, (d/b)*n_b must be an integer
    # check that (d*n_b)%b == 0
        try:
            hold = int(input('\n\nEnter n_b: '))
            while ((hold * out[3]) % out[2] != 0):
                print('\n\nERROR: Entered n_b does not place the strip in the mesh')
                hold = int(input('\n\nEnter n_b: '))
            while (hold < 1):
                print('\n\nERROR: n_b must be positive')
                hold = int(input('\n\nEnter n_b: '))
            out.append(hold)
            break
        except ValueError:
            print('\n\nERROR: n_b must be an integer')
    #
    # Get epsilon_r
    while (True):
        try:
            hold = float(input('\n\nEnter epsilon_r: '))
            while (hold <= 0):
                print('\n\nERROR: epsilon_r must be positive')
                hold = float(input('\n\nEnter epsilon_r: '))
            out.append(hold)
            break
        except ValueError:
            print('\n\nERROR: epsilon_r must be a number')
    #
    # Check if a 3d plot is to be generated
    hold = input('\n\nWould you like to to generate and display a 3-D plot of the potential distribution on the specified grid? (y/n):
')
    while (hold != 'Y' and hold != 'y' and hold != 'N' and hold != 'n'):
        print('\n\nERROR: Please answer (y/n)')
        hold = input('\n\nWould you like to to generate and display a 3-D plot of the potential distribution on the specified grid?
(y/n): ')
    if hold == 'Y' or hold == 'y':
        out.append(True)
    else:
        out.append(False)
    #
    # Check if the strip is to have finite thickness k.
```

```
    hold = input('\n\nWould you like to model a strip of finite thickness k? (y/n): ')
    while (hold != 'Y' and hold != 'y' and hold != 'N' and hold != 'n'):
        print('\n\nERROR: Please answer (y/n)')
        hold = input('\n\nWould you like to model a strip of finite thickness k? (y/n): ')
    if hold == 'Y' or hold == 'y':
        out.append(True)
    else:
        out.append(False)
    print('\n\n')
    #
    return out


# main
para = getUserInput()
#test cases
#para = [3,1,2,1,3,1,60,60,1,True,False]
#para = [3,1,2,1,3,1,120,120,1,False,False]
#para = [3,1,2,1,3,1,240,240,1,False,False]
#para = [3,1,2,1,3,1,480,480,1,False,False]
#Line2
#para = [3,1,10,1,3,1,60,60,1,True,False]
#para = [3,1,10,1,3,1,120,120,1,False,False]
#para = [3,1,10,1,3,1,240,240,1,False,False]
#para = [3,1,10,1,3,1,480,480,1,False,False]
#Line3
#para = [3,1,10,1,3,1,60,60,10,True,False]
#para = [3,1,10,1,3,1,120,120,10,False,False]
#para = [3,1,10,1,3,1,240,240,10,False,False]
#para = [3,1,10,1,3,1,480,480,10,False,False]
#Bonus Finite
#para = [5,12,25,12,10,1,50,25,2.3,True, True]
#para = [5,12,25,12,10,1,50,25,1,False, True]



f=para[10]
alpha = (para[7]/para[6])*(para[4]/para[5])*(para[0]/para[1])*(para[3]/para[2])
n_a = para[6]
n_b = para[7]
omega_opt = 2 * (1-(pi/sqrt(2))*sqrt((1/(n_a**2))+(1/(n_b**2))))
tol = 10**(-5)
strip_height = (para[3]*n_b)//para[2]
strip_width = (para[5]*n_a)//para[4]
# indexing: [row][column] with row 0 at the lower boundary in the
# substrate region and column zero at the left side of the cutout
potential = []
for i in range(n_b+1):
    potential.append([])
    for j in range(n_a+1):
        potential[i].append(0)
        if((i == strip_height and j <= strip_width) or (i == (strip_height+int(f)) and j <= strip_width)):
            potential[i][j] = 1
residual = []
for i in range(n_b+1):
```

```
    residual.append([])
    for j in range(n_a+1):
        residual[i].append(None)


#iteration
rmax=1
iterations=0
while(rmax > tol):
    rmax=0
    if(iterations==1000):
        break
    for i in range(1,len(potential)-1):
        for j in range(len(potential[i])-1):
            if((i == strip_height and j <= strip_width) or (i == (strip_height+int(f)) and j <= strip_width)):
                continue
            if(i == strip_height):
                residual[i][j]=calc_residual(potential,i,j,alpha,para[8])
            else:
                residual[i][j]=calc_residual(potential,i,j,alpha)
            if(abs(residual[i][j])>rmax):
                rmax=abs(residual[i][j])
            potential[i][j] += omega_opt*residual[i][j]
    iterations+=1
print('\n\n')
print('Final Maximum Residual: '+str(rmax))
print('\n\n')
print('Total Iterations: '+str(iterations))
print('\n\n')
print('Calculated Capacitance: '+str(capacitance(potential,strip_width,strip_height,alpha,para[8])))
print('\n\n')
print('Omega_opt: '+str(omega_opt))
print('\n\n')
if(para[9]):
    plot(potential)
```

Appendix P2: Raw Output Screenshots)
Line 1) (N=60, 120, 240, 480)

```
Final Maximum Residual: 9.501271313827608e-06

Total Iterations: 99

Calculated Capacitance: 13.9230767718528

Omega_opt: 1.8952802448803403
```

```
Final Maximum Residual: 9.969671784793022e-06

Total Iterations: 206

Calculated Capacitance: 13.84298468919578

Omega_opt: 1.94764012244017
```

```
Final Maximum Residual: 9.877948213776744e-06

Total Iterations: 421

Calculated Capacitance: 13.803831172910812

Omega_opt: 1.9738200612200851
```

```
Final Maximum Residual: 9.980451407209934e-06

Total Iterations: 848

Calculated Capacitance: 13.784453341762045

Omega_opt: 1.9869100306100425
```

Line 2) (N=60, 120, 240, 480)

```
Final Maximum Residual: 9.643209820775489e-06

Total Iterations: 170

Calculated Capacitance: 9.072989789429332

Omega_opt: 1.8952802448803403
```

```
Final Maximum Residual: 9.83144144783843e-06

Total Iterations: 292

Calculated Capacitance: 9.008898671578677

Omega_opt: 1.94764012244017
```

```
Final Maximum Residual: 9.940175110190186e-06

Total Iterations: 506

Calculated Capacitance: 8.979585689914343

Omega_opt: 1.9738200612200851
```

```
Final Maximum Residual: 9.978115500797191e-06

Total Iterations: 875

Calculated Capacitance: 8.969430063700194

Omega_opt: 1.9869100306100425
```

Line 3) (N=60, 120, 240, 480)

```
Final Maximum Residual: 9.95526574509853e-06

Total Iterations: 161

Calculated Capacitance: 72.21689937628472

Omega_opt: 1.8952802448803403
```

```
Final Maximum Residual: 9.902147162066388e-06

Total Iterations: 277

Calculated Capacitance: 71.62737172176463

Omega_opt: 1.94764012244017
```

```
Final Maximum Residual: 9.927465488057674e-06

Total Iterations: 481

Calculated Capacitance: 71.34026027827996

Omega_opt: 1.9738200612200851
```

```
Final Maximum Residual: 9.949806875853007e-06

Total Iterations: 856

Calculated Capacitance: 71.20068370181154

Omega_opt: 1.9869100306100425
```

Finite Strip) (substrate line, then air-line)

```
Final Maximum Residual: 9.97413466991004e-06

Total Iterations: 50

Calculated Capacitance: 5.093205084735854

Omega_opt: 1.801308234684078
```

```
Final Maximum Residual: 9.897446653223119e-06

Total Iterations: 52

Calculated Capacitance: 2.8077175895682385

Omega_opt: 1.801308234684078
```