FDM-devnote

February 27, 2020

1 Finite difference method :

The following notebook uses the implemented functions to set up and solve an electrostatic and current static problem with the finite difference method. The implemented functions can be found in the *fdm_funs.py* file. The functions implemented are :

- **init_problem_1(...)** : initializes the electrostatic problem by generating nodes for the selected dimensions and sets the prescribed boundary values.
- **init_problem_2(...)** : initializes the current static problem by generating nodes for the selected dimensions and sets the prescribed boundary values.
- **solve(..,)** : calculates the potential and field values and returns them
- **plot_functions**

To get *interactive plots* please uncomment the last line in the following code block.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import fdm_funs as fd

## Uncomment the following line to get interactive plots :

# %matplotlib notebook
```

1.1 Problem 1. (Electrostatic case)

For the given problem it is known that the length L_A in the $\hat{z}$ direction is much bigger than the length L in the $\hat{x}$ and $\hat{y}$ directions. Therefore the the derivative of the potential along the z-axis is zero and the Laplace equation becomes :

$\Delta V = \frac{\delta^2 V}{\delta x^2} + \frac{\delta^2 V}{\delta y^2} = 0.$

No Neumann boundary conditions were given for this problem and therefore : $\frac{\delta V}{\delta n} = 0$, Dirichlet boundary conditions were prescribed in form of Potential values at defined edges with different values for cases a), b) and c).

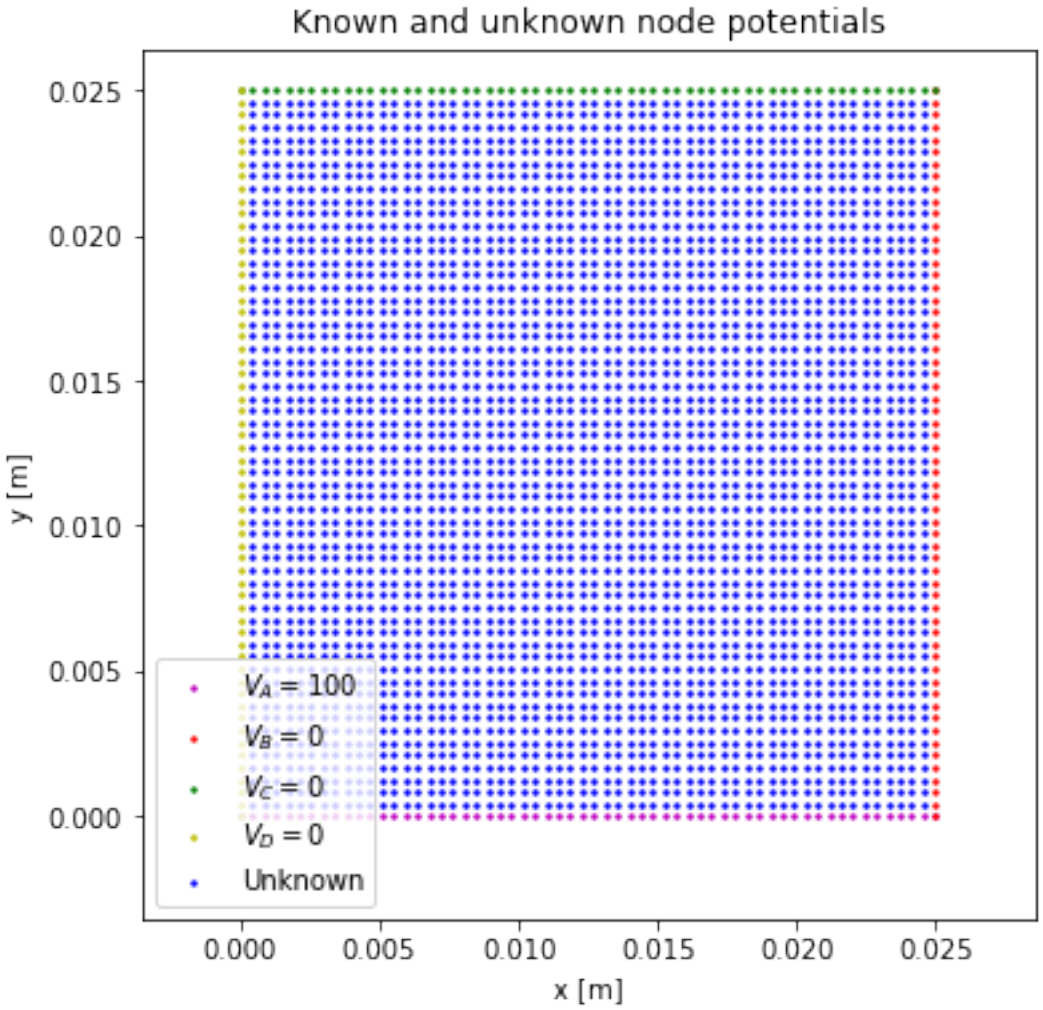
1.1.1 Initializing problem for a) :

Initializing the problem to solve for the unknown potentials with the given potentials (Dirichlet boundary values) at the four edges : - $V_A = 100$ - $V_B = 0$ - $V_C = 0$ - $V_D = 0$

and setting $N = 60$ which gives us $d = \frac{L}{N} = \frac{2 \cdot 10^{-2}}{40} = 0.333 \cdot 10^{-4}$ which is the step size from one node to the other, setting N to a larger number gives us more precise solution because we get a finer discretization of the problem geometry but this leads to more computations which need to be performed.

For the initialization of the first problem a mesh of points is generated with the selected length, width and stepsize. Then for this mesh we assign the prescribed boundary values to the nodes at the edges using logical indexing and choosing the desired region. And then return a matrix **V** containing the node potentials, a matrix **I** which contains the information which nodes have potentials assigned to them and matrices **xx** and **yy** which contain the mesh coordinates.

[5]:
```
N = 60
L = 2.5e-2                                  #   Va  Vb Vc Vd
V, I, xx, yy = fd.init_problem_1(N, L, Vp=[100, 0, 0, 0], plot_nodes=True)
```



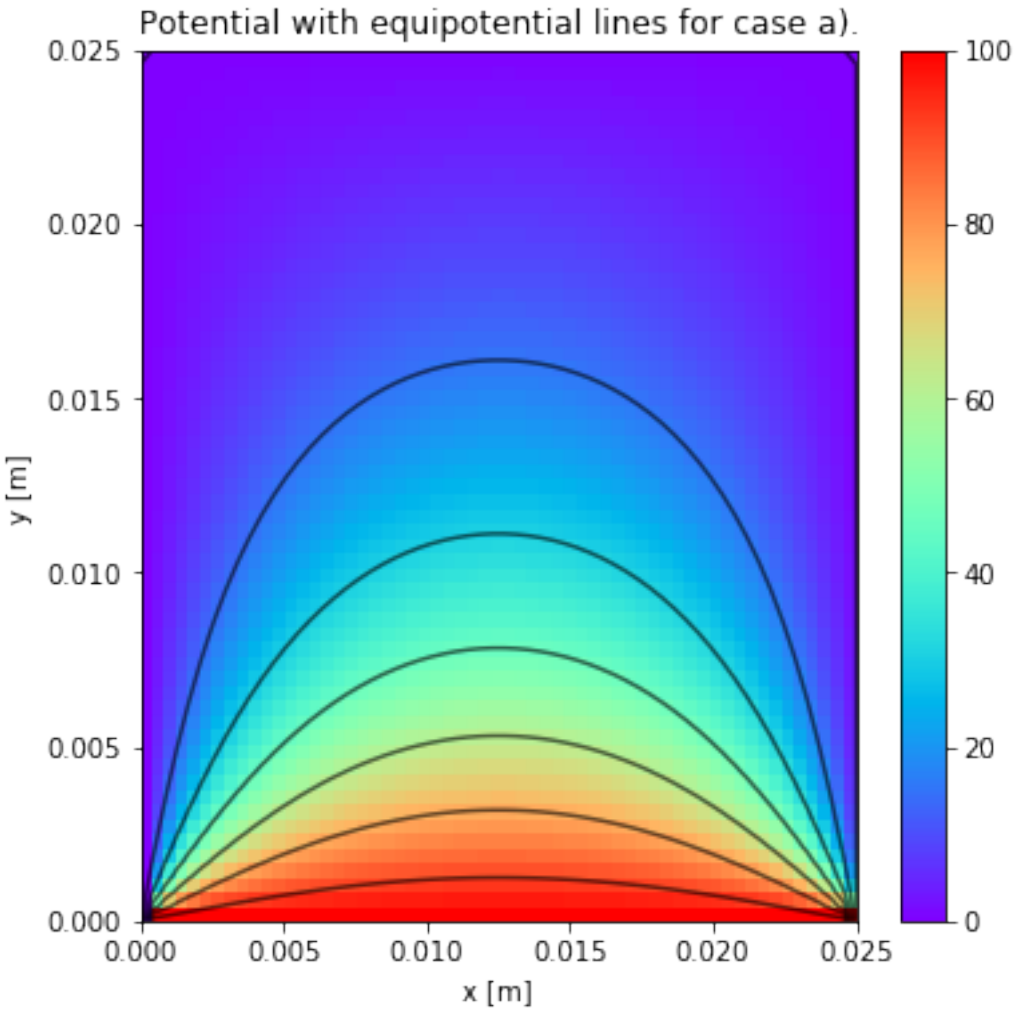
The plot above shows the nodes and the associated potential values. The blue nodes are the ones that need to be calculated and therefore are marked as unknown.

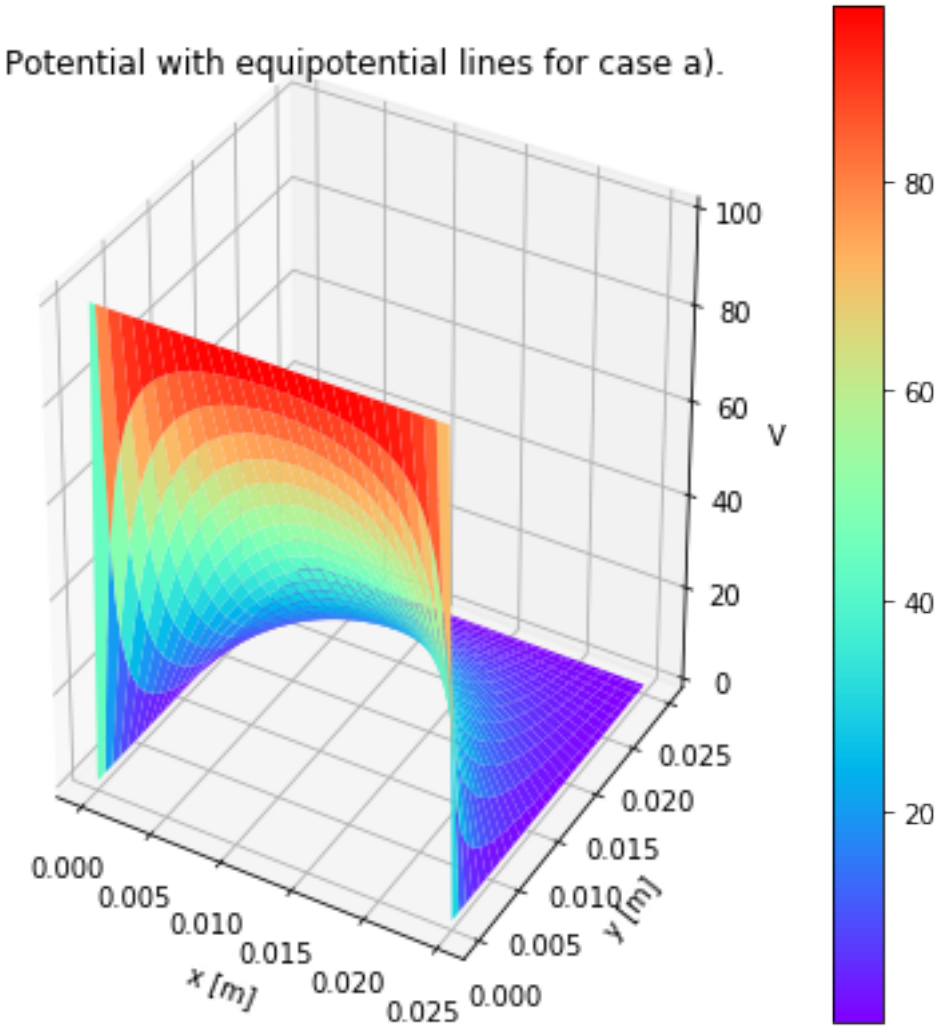
1.1.2 Solving

To get the solution (the unknown potentials) the solve function implements the finite difference algorithm. It goes over the unknown nodes and for every node sets the values of the stiffnes matrix **K** and the resulting vector **Vk** which we need to solve for the unknown potentials **V**: **KV** = **Vk**. The resulting matrix **K** is a bandmatrix, which means that it is a sparse matrix (most of its values are zero) and the ones which are not are located at and around the main diagonal. So for storing these kind of matrices for big problems sparse matrix containers are used which save only the nonzero values and use up much less memory but for our purpose full matrices are acceptable.

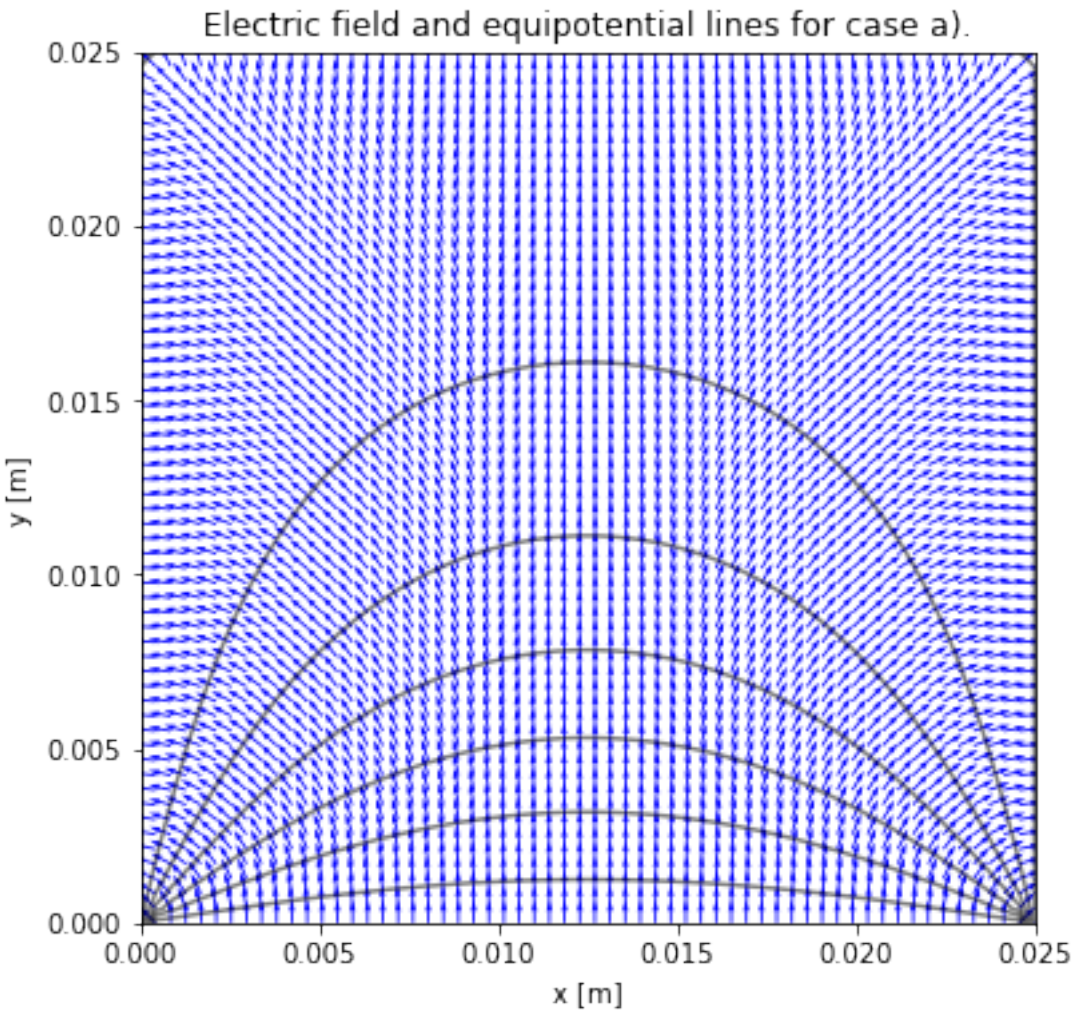
The other part is the calculation of the field values, we know that : $E = -\nabla V = -[\frac{\delta V}{\delta x} \quad \frac{\delta V}{\delta y}]$. To numerically compute the gradient we need to compute the differences of the potentials along the x and along the y directions. Because we have evenly spaced data points the partial derivatives can be computed as $f_i = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$.

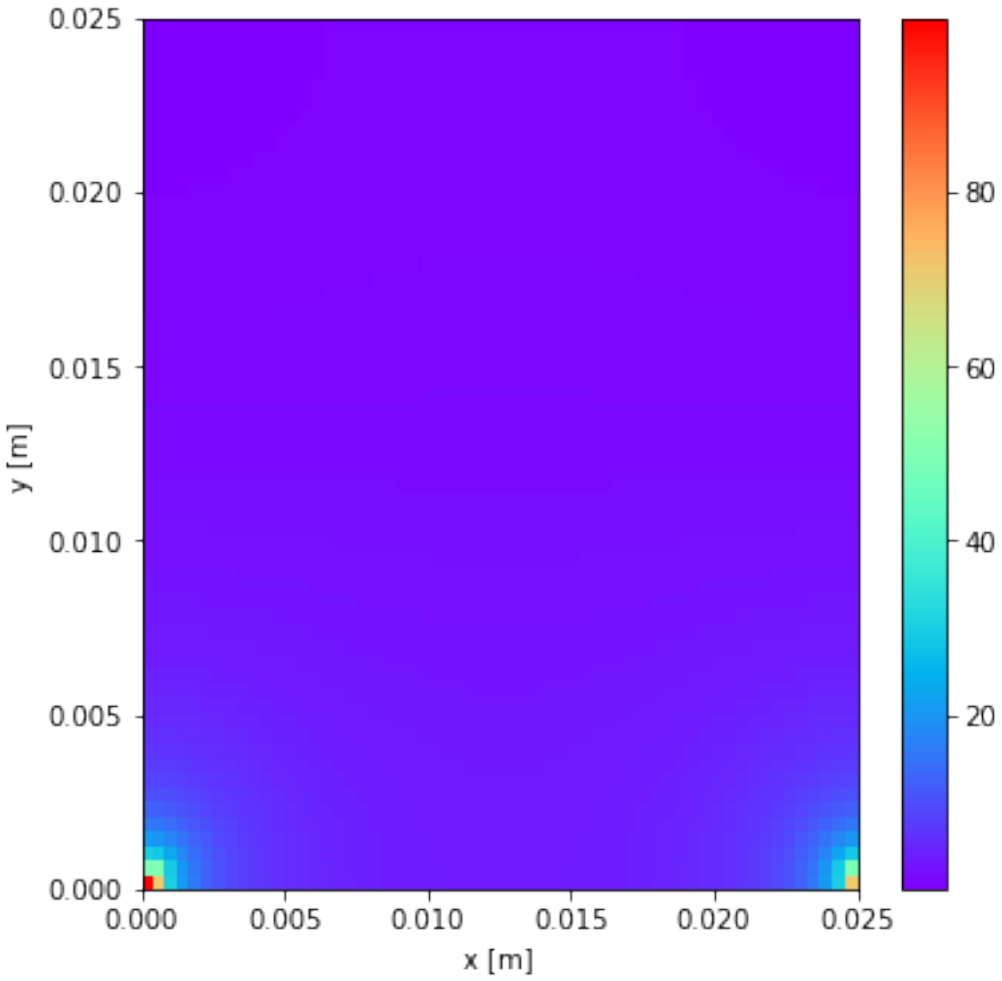
[6]:
```
V, Ex, Ey = fd.solve(N, V, I)
fd.plot_potential(V,xx,yy,titl="Potential with equipotential lines for case a).
 ↪")
fd.plot_field(Ex,Ey,V,xx,yy,titl="Electric field and equipotential lines for␣
 ↪case a).")
```


Potential with equipotential lines for case a).
0.025
0.020
0.015
0.010
0.005
0.000
y [m]
0.000
0.005
0.010
0.015
0.020
0.025
x [m]
100
80
60
40
20
0




```
/home/jpajser/gits/FDM-solver/FDM-solver/fdm_funs.py:71: RuntimeWarning: invalid
value encountered in true_divide
  Ex = Ex/E
/home/jpajser/gits/FDM-solver/FDM-solver/fdm_funs.py:72: RuntimeWarning: invalid
value encountered in true_divide
  Ey = Ey/E
```


Electric field and equipotential lines for case a).
0.025
0.020
0.015
0.010
0.005
0.000
y [m]
0.000
0.005
0.010
0.015
0.020
0.025
x [m]



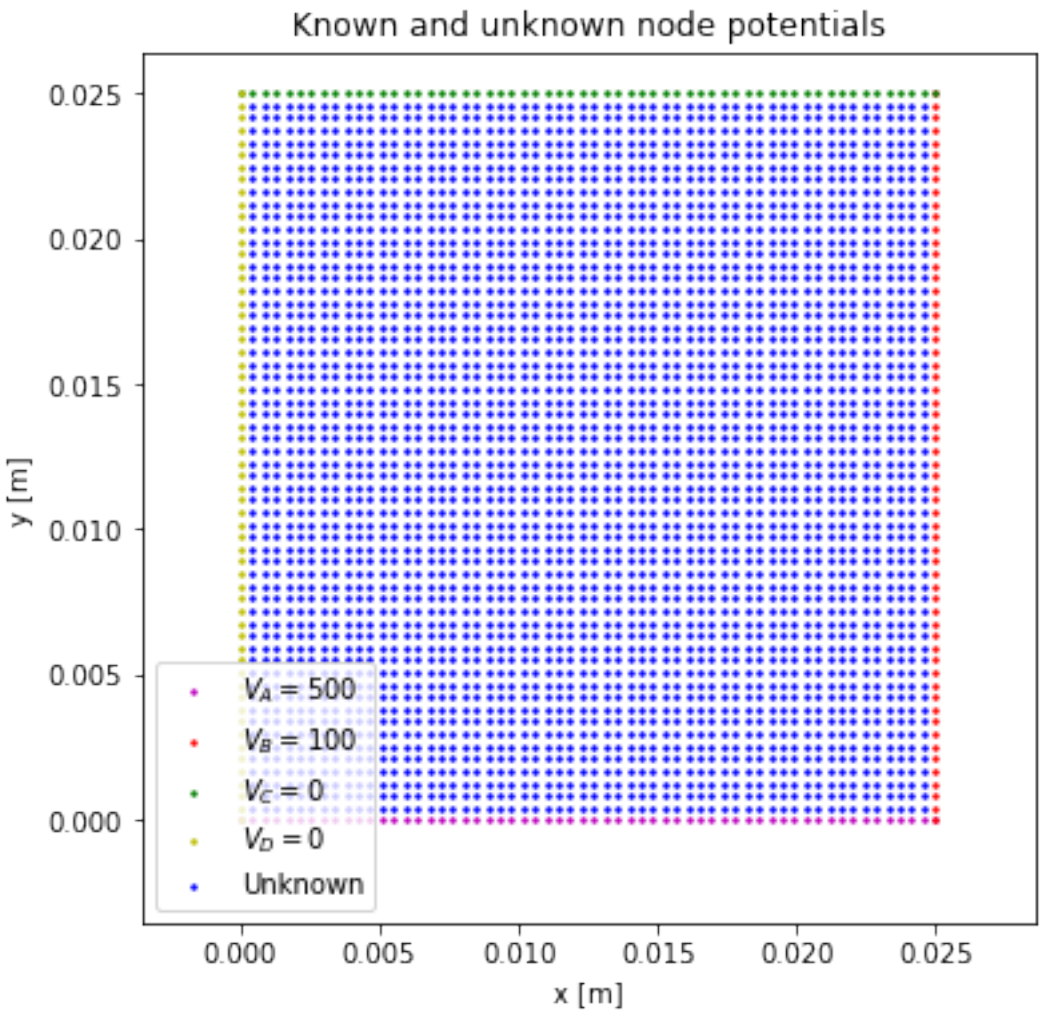
The field vectors were normalized to get a nicer plot where we can see the direction of the field in all the regions.

We should note the corner points in the field plot where singularities arise and the field has a much larger magnitude than in the other nodes due to the locations of the prescribed boundary values.

1.1.3 Initializing problem for b) :

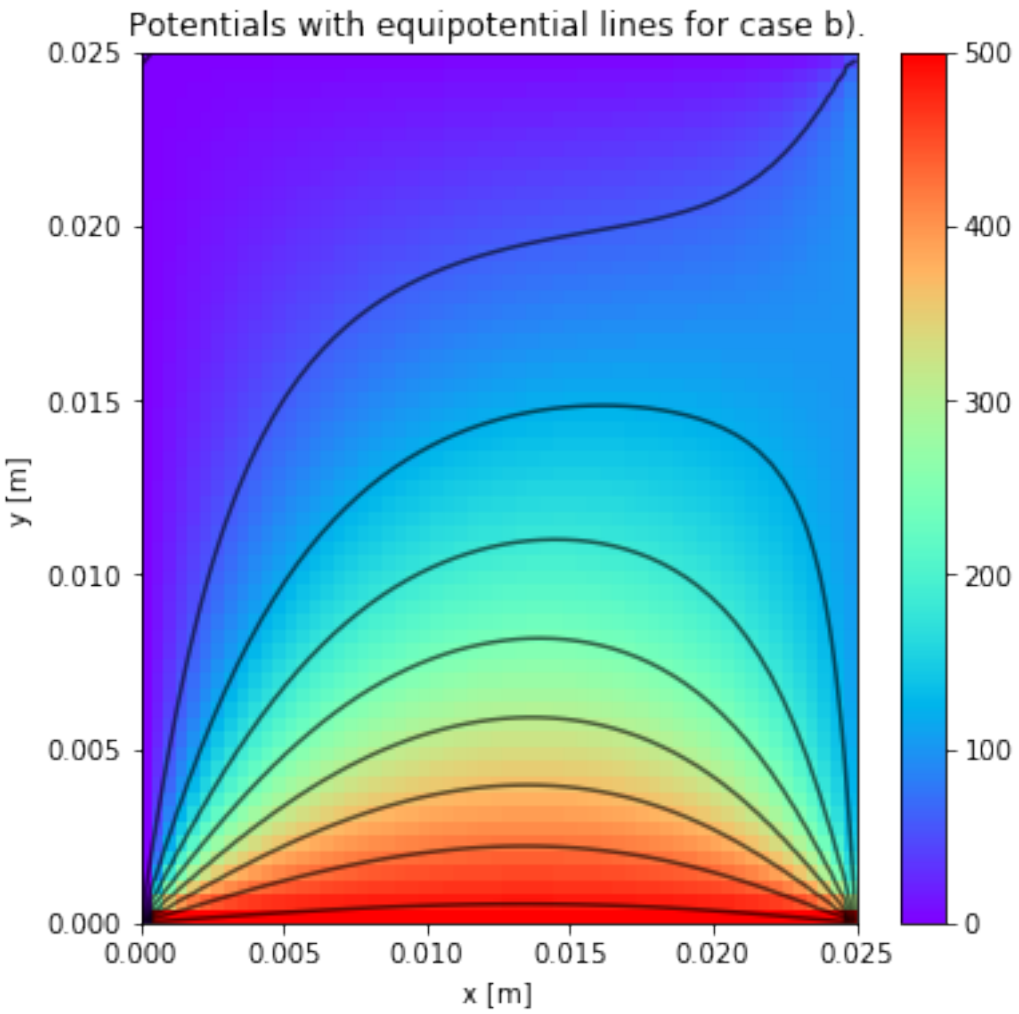
Initializing the problem to solve for the given potentials : - $V_A = 500$ - $V_B = 100$ - $V_C = 0$ - $V_D = 0$ with the same discretization as before.

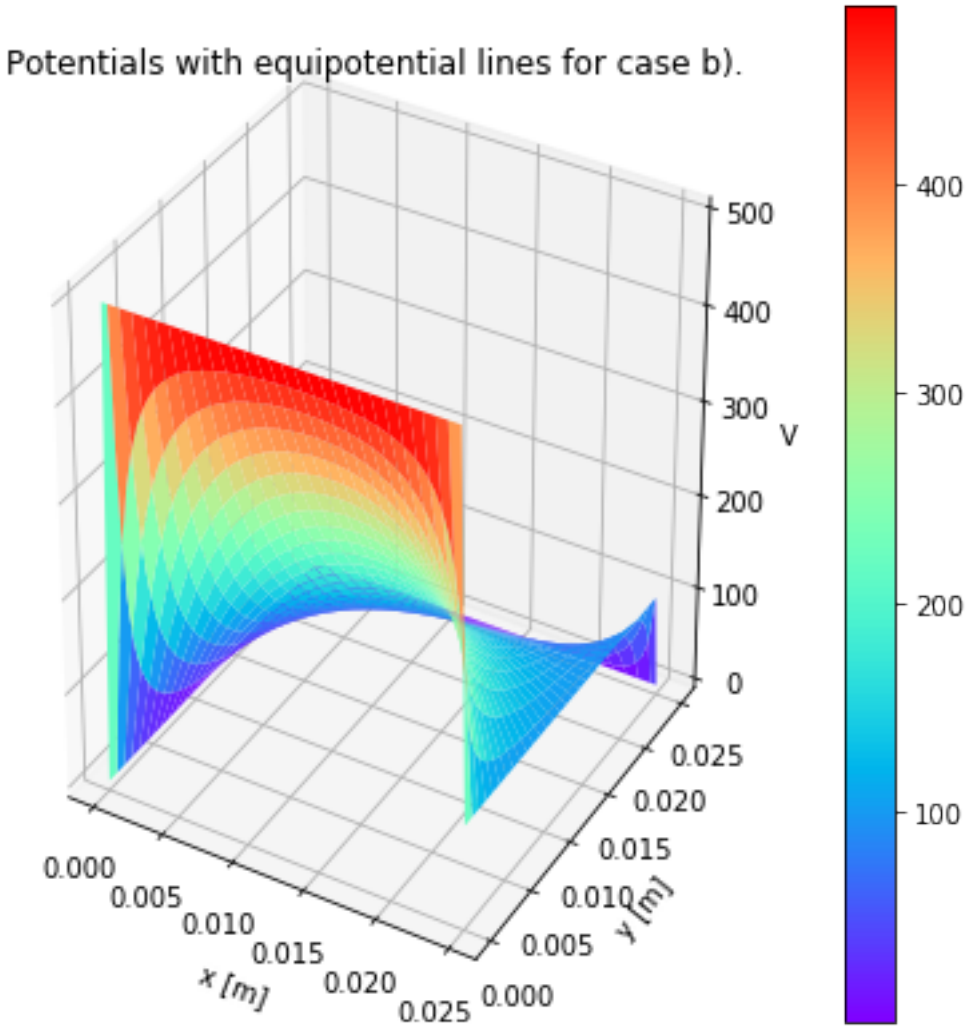
[7]:
```
V, I, xx, yy = fd.init_problem_1(N, L, Vp=[500,100,0,0], plot_nodes=True)
```

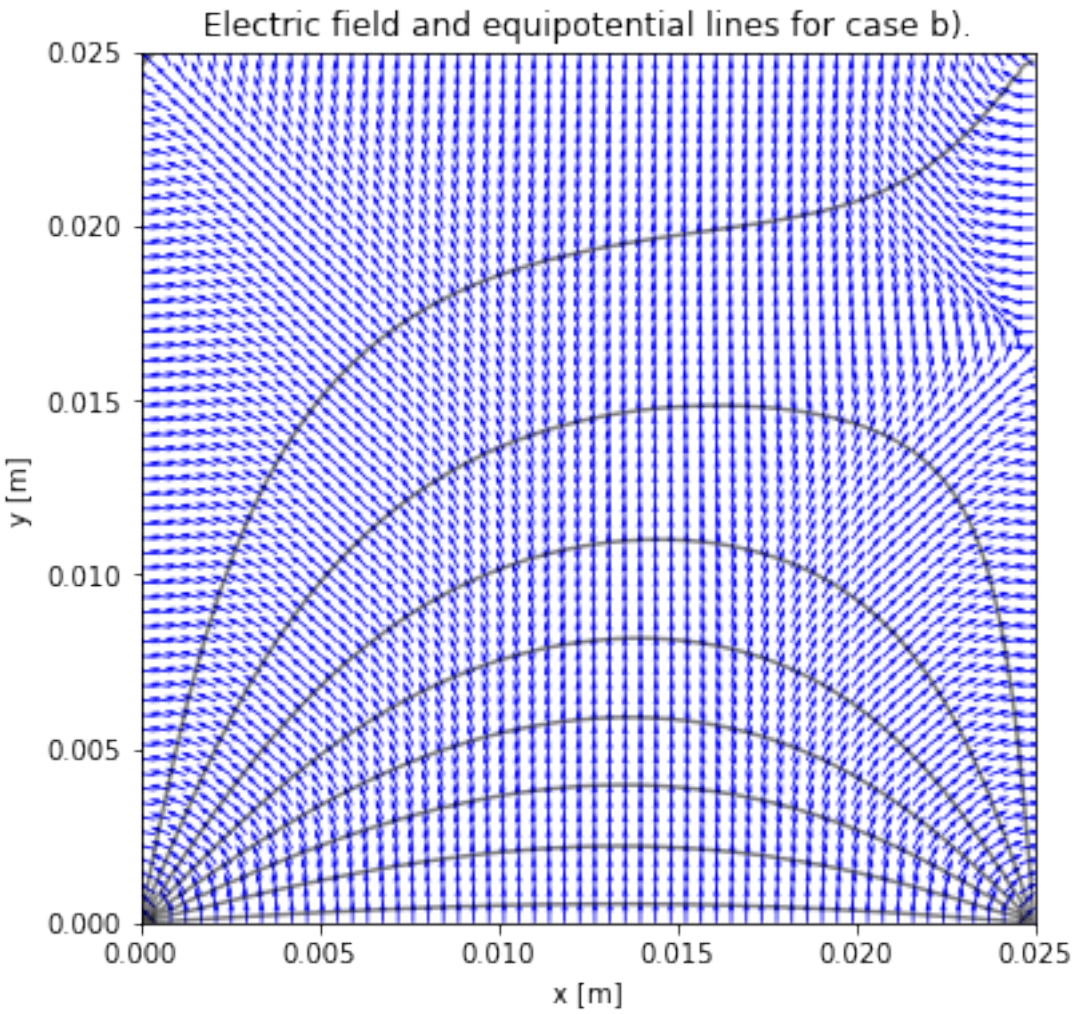


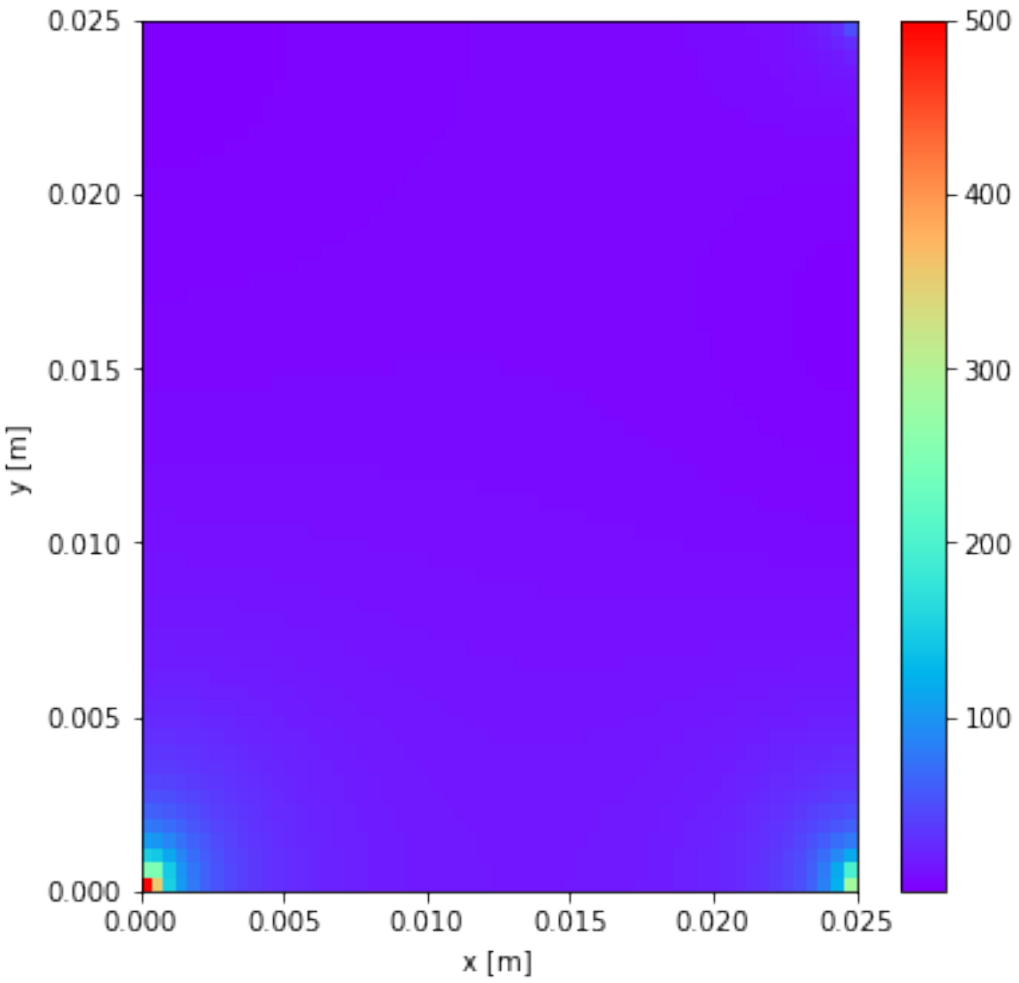
1.1.4 Solution :

[8]:
```
V, Ex, Ey = fd.solve(N, V, I)
fd.plot_potential(V,xx,yy,titl="Potentials with equipotential lines for case b).
 ↪")
fd.plot_field(Ex,Ey,V,xx,yy,titl="Electric field and equipotential lines for␣
 ↪case b).")
```


Potentials with equipotential lines for case b).
0.025
0.020
0.015
0.010
0.005
0.000
y [m]
0.000
0.005
0.010
0.015
0.020
0.025
x [m]
500
400
300
200
100
0


Potentials with equipotential lines for case b).
500
400
300
200
100
0
V
0.000
0.005
0.010
0.015
0.020
0.025
x [m]
0.000
0.005
0.010
0.015
0.020
0.025
y [m]
400
300
200
100


Electric field and equipotential lines for case b).
0.025
0.020
0.015
0.010
0.005
0.000
y [m]
0.000
0.005
0.010
0.015
0.020
0.025
x [m]

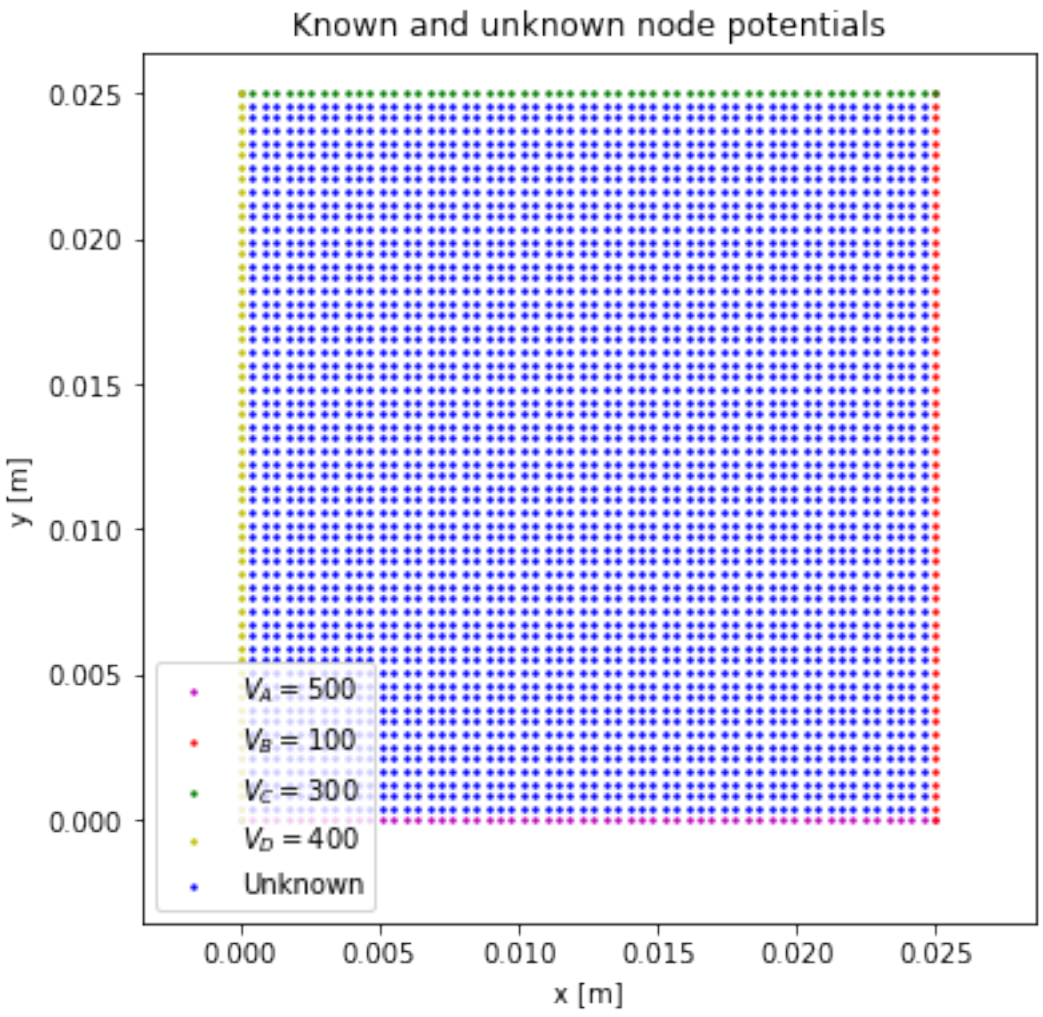


1.1.5 Initializing problem for c) :

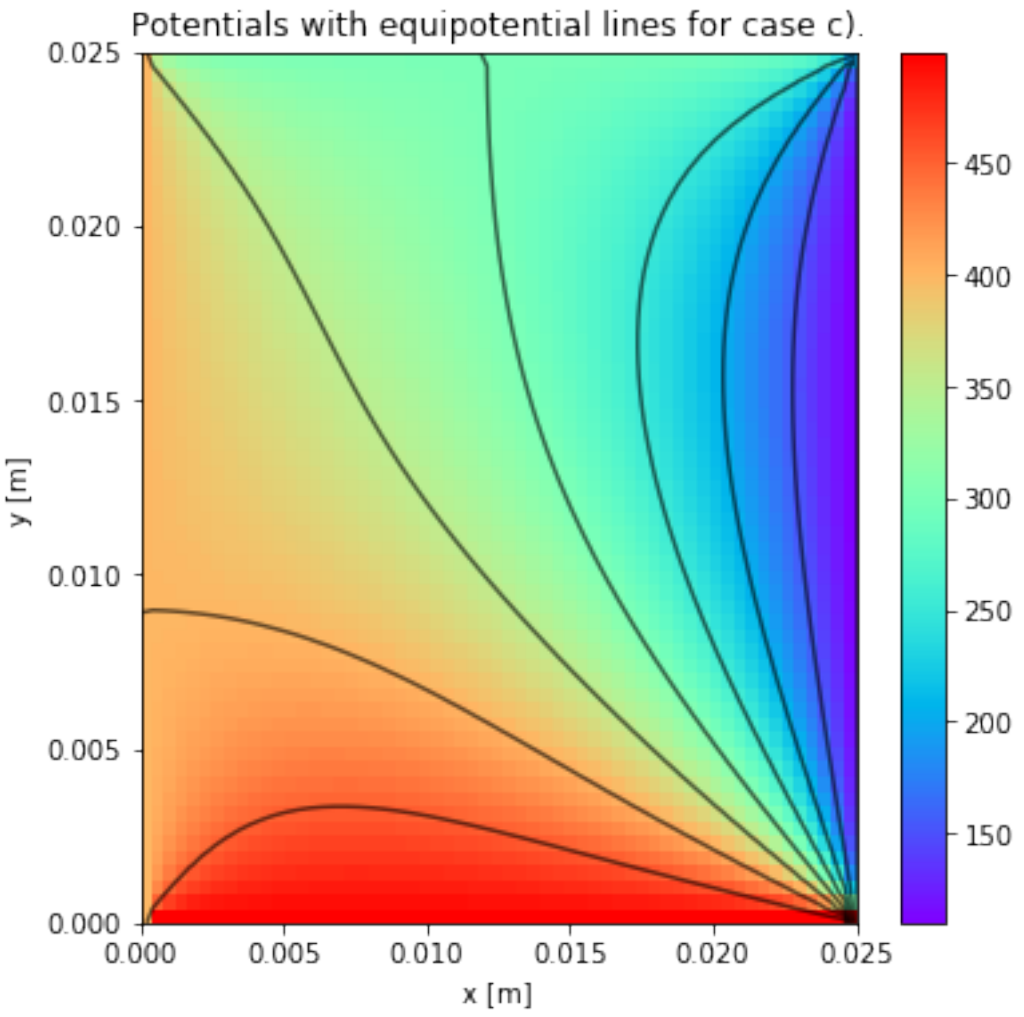
Initializing the problem to solve for the given potentials : - $V_A = 500$ - $V_B = 100$ - $V_C = 300$ - $V_D = 400$

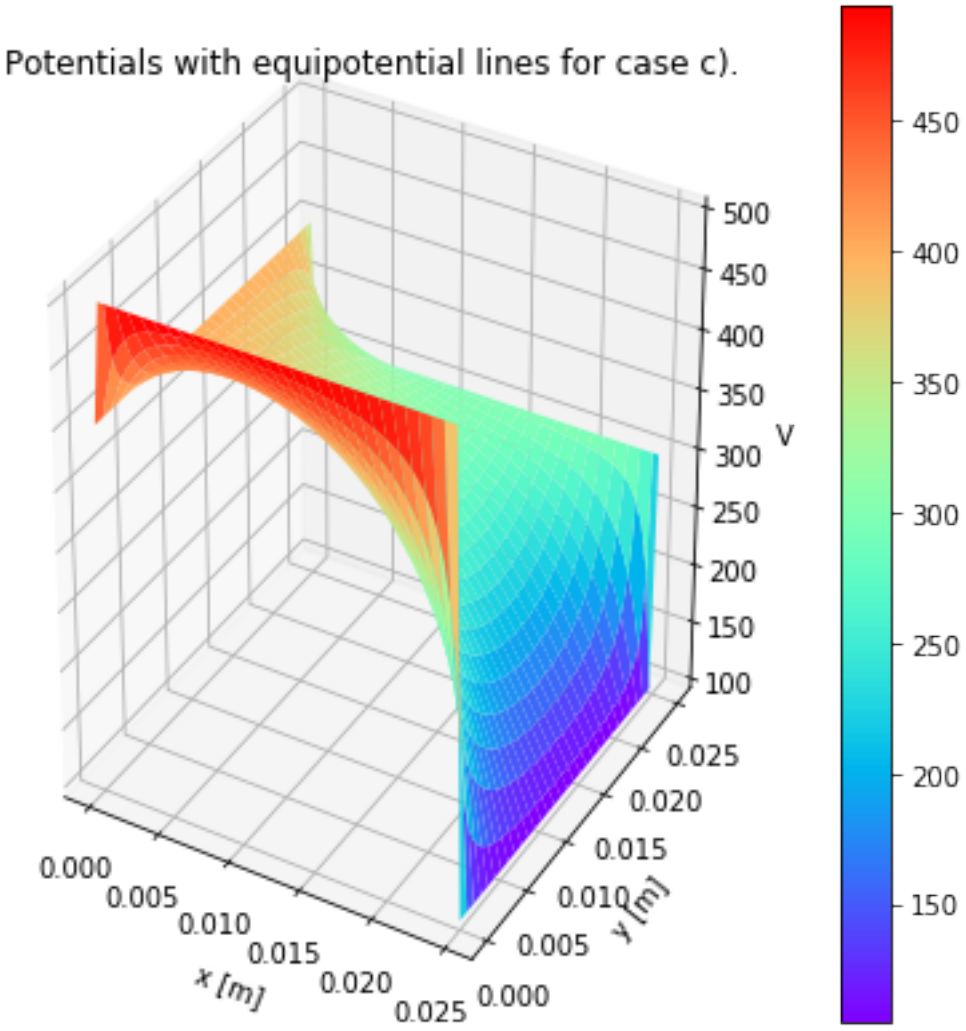
with the same discretization as before.

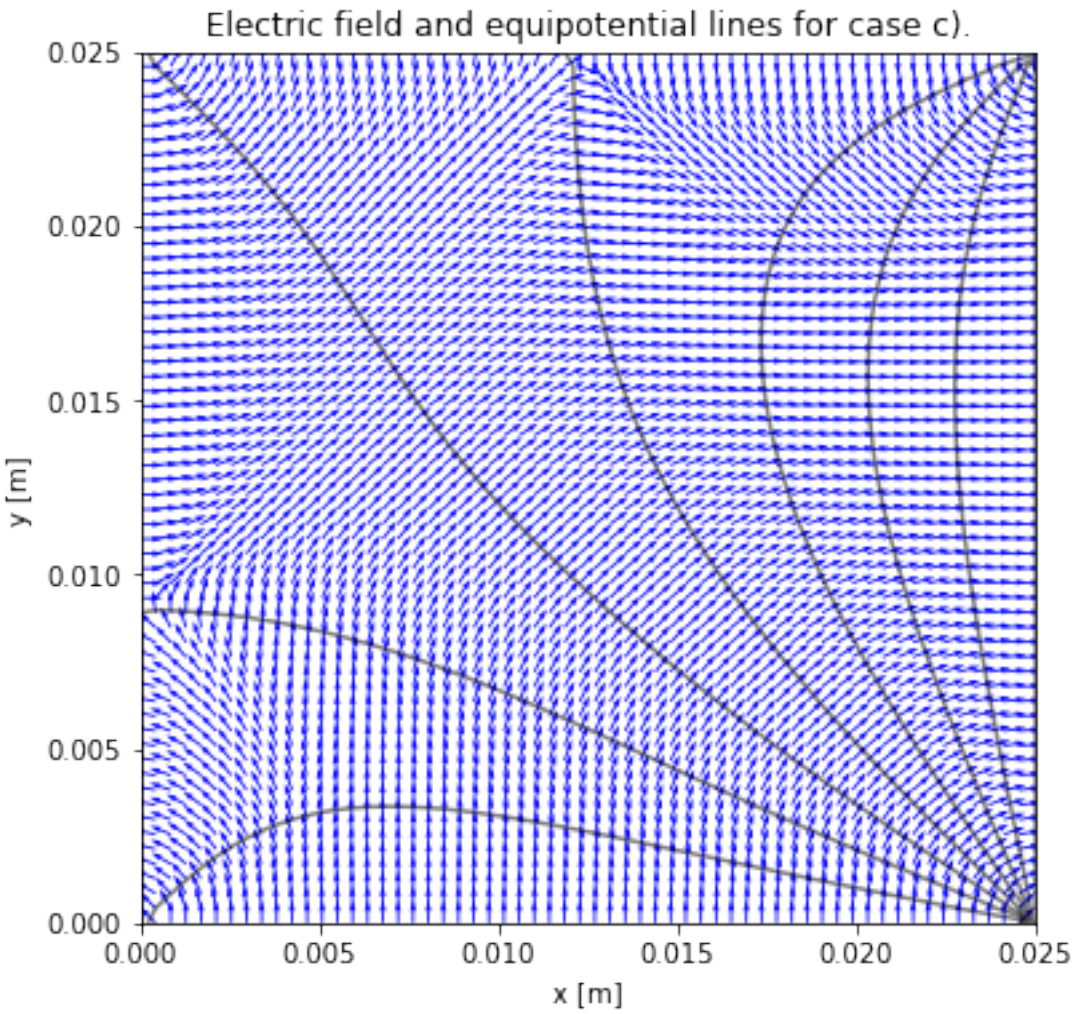
[9]:
```
V, I, xx, yy = fd.init_problem_1(N, L, Vp=[500,100,300,400], plot_nodes=True)
```

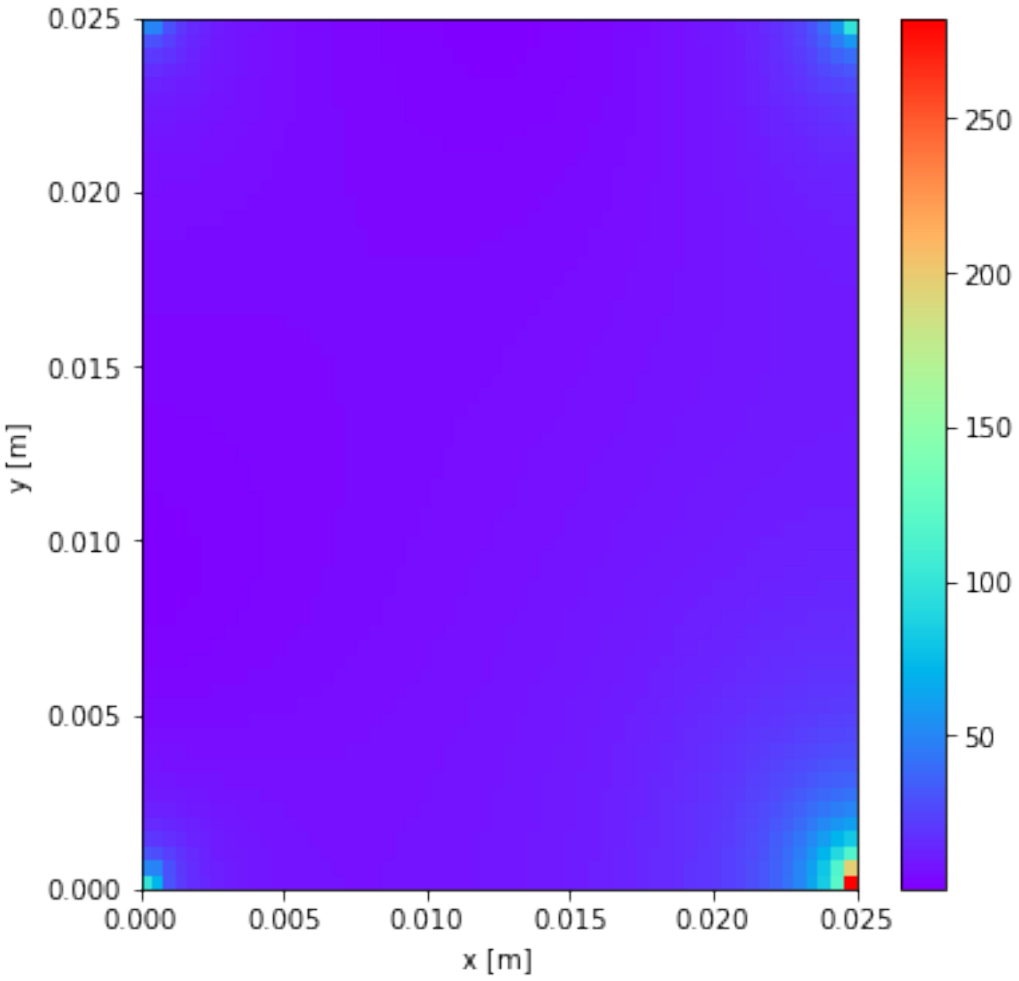


[10]:
```
V, Ex, Ey = fd.solve(N, V, I)
fd.plot_potential(V,xx,yy,titl="Potentials with equipotential lines for case c).
 ↪")
fd.plot_field(Ex,Ey,V,xx,yy,titl="Electric field and equipotential lines for␣
 ↪case c).")
```


Potentials with equipotential lines for case c).
0.025
0.020
0.015
0.010
0.005
0.000
y [m]
0.000
0.005
0.010
0.015
0.020
0.025
x [m]
450
400
350
300
250
200
150


Potentials with equipotential lines for case c).
500
450
400
350
300
250
200
150
100
V
0.000
0.005
0.010
0.015
0.020
0.025
x [m]
0.000
0.005
0.010
0.015
0.020
0.025
y [m]
450
400
350
300
250
200
150


Electric field and equipotential lines for case c).
0.025
0.020
0.015
0.010
0.005
0.000
y [m]
0.000
0.005
0.010
0.015
0.020
0.025
x [m]



The equipotential lines are lines of constant potential and they are plotted using the contour function which just plots the field lines at the same magnitude level.

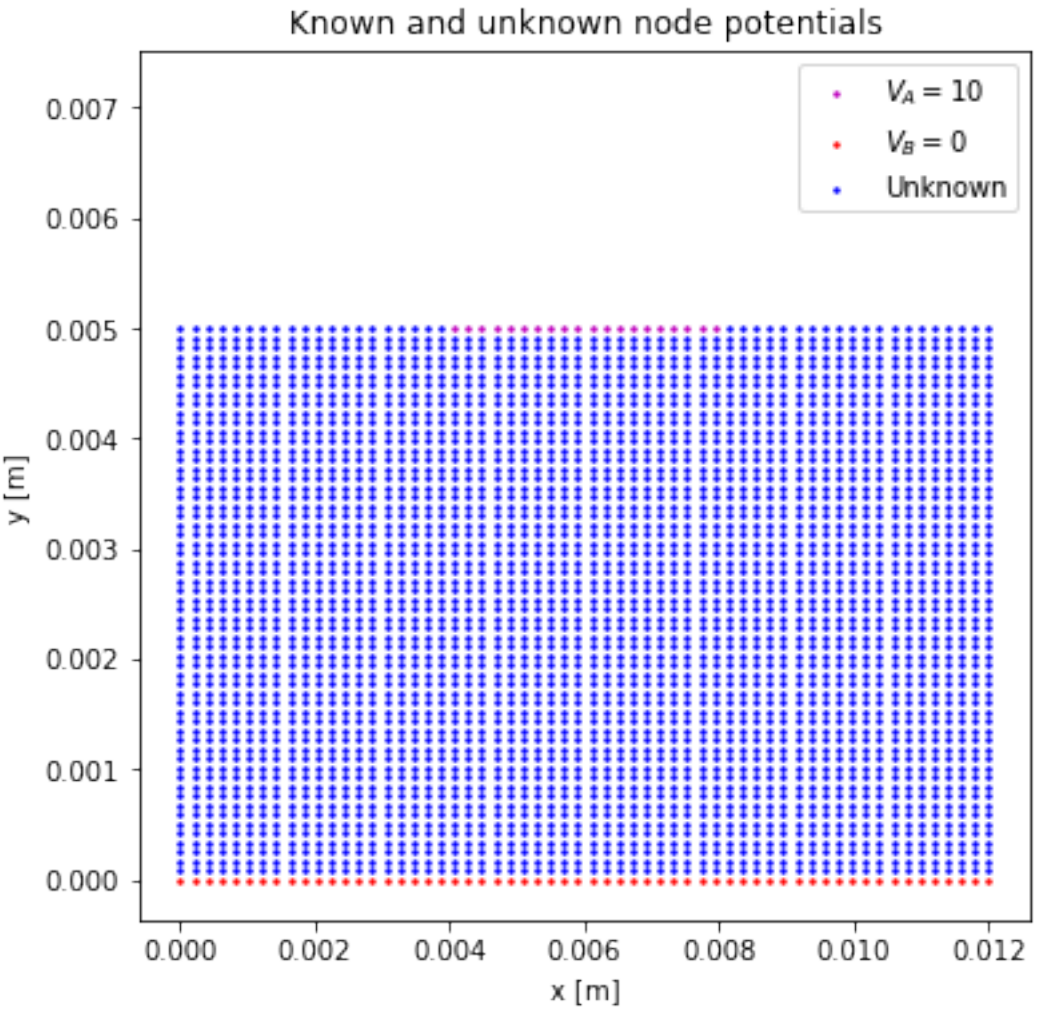
1.2 Problem 2. (Stationary current)

The initialization function for this problem is almost the same as in the first problem with the only exception beeing a different selection of nodes which get the prescribed boundary values. In this case we also had no Neumann boundary conditions and two edges with dirichlet boundary conditions. The potentials and field values are calculated by the solve function and plotted below.

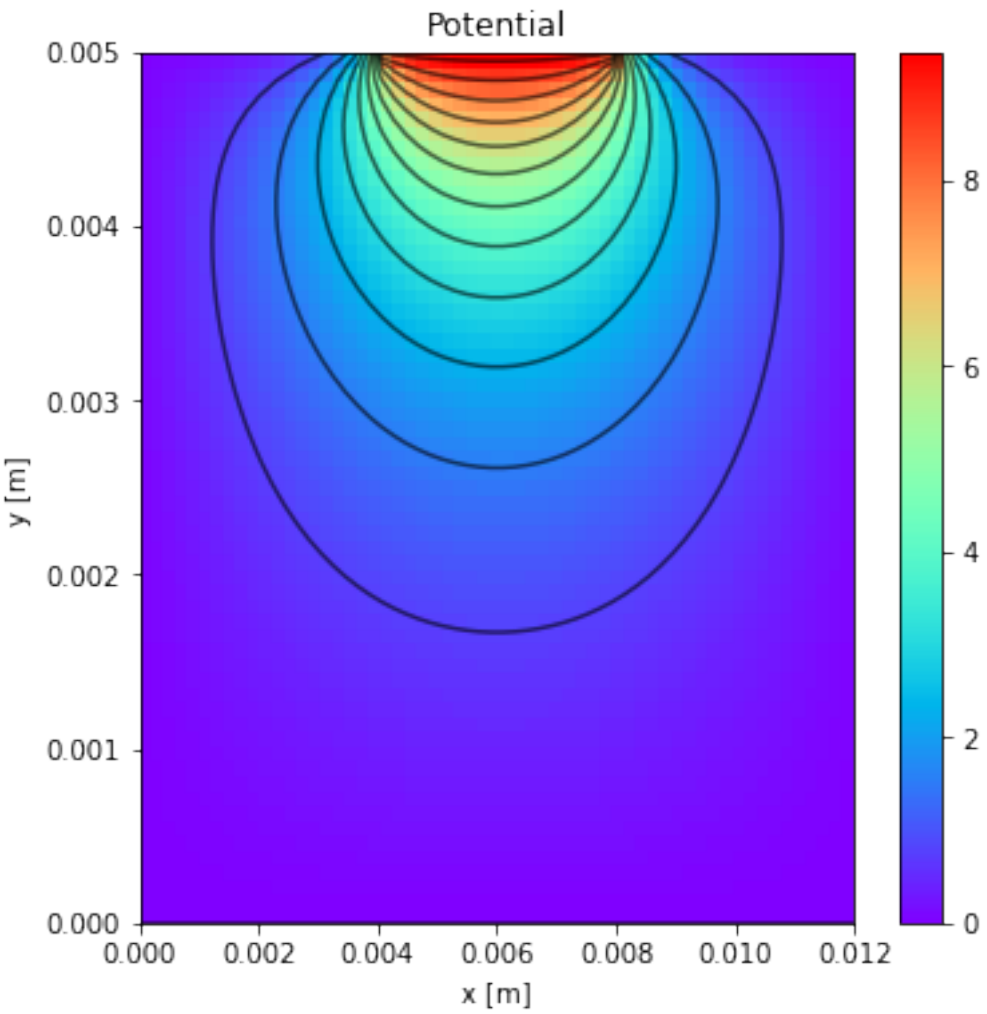
It is worth noticing a similar behaviour as in the first problem, where the field magnitude at the ends of the prescribed boundary goes up very quickly to a very large value.

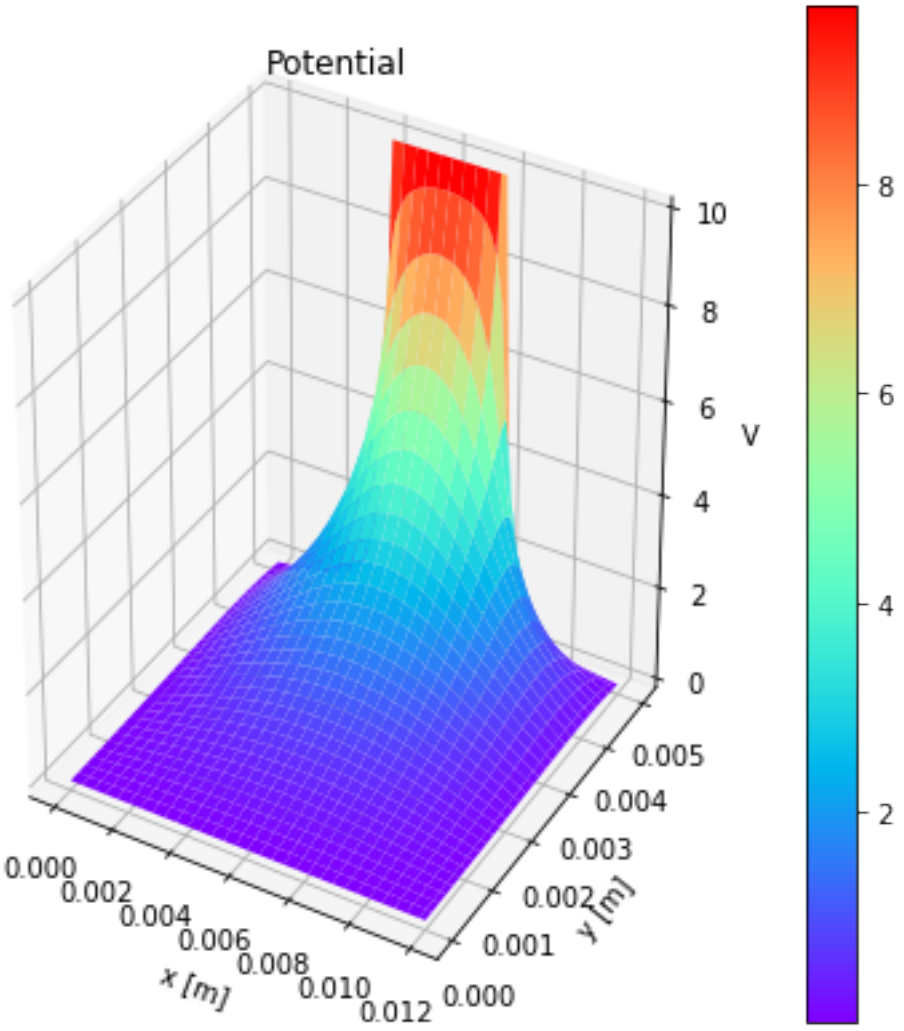
[11]:
```
N = 60
Lx = 12e-3
Ly = 5e-3
```

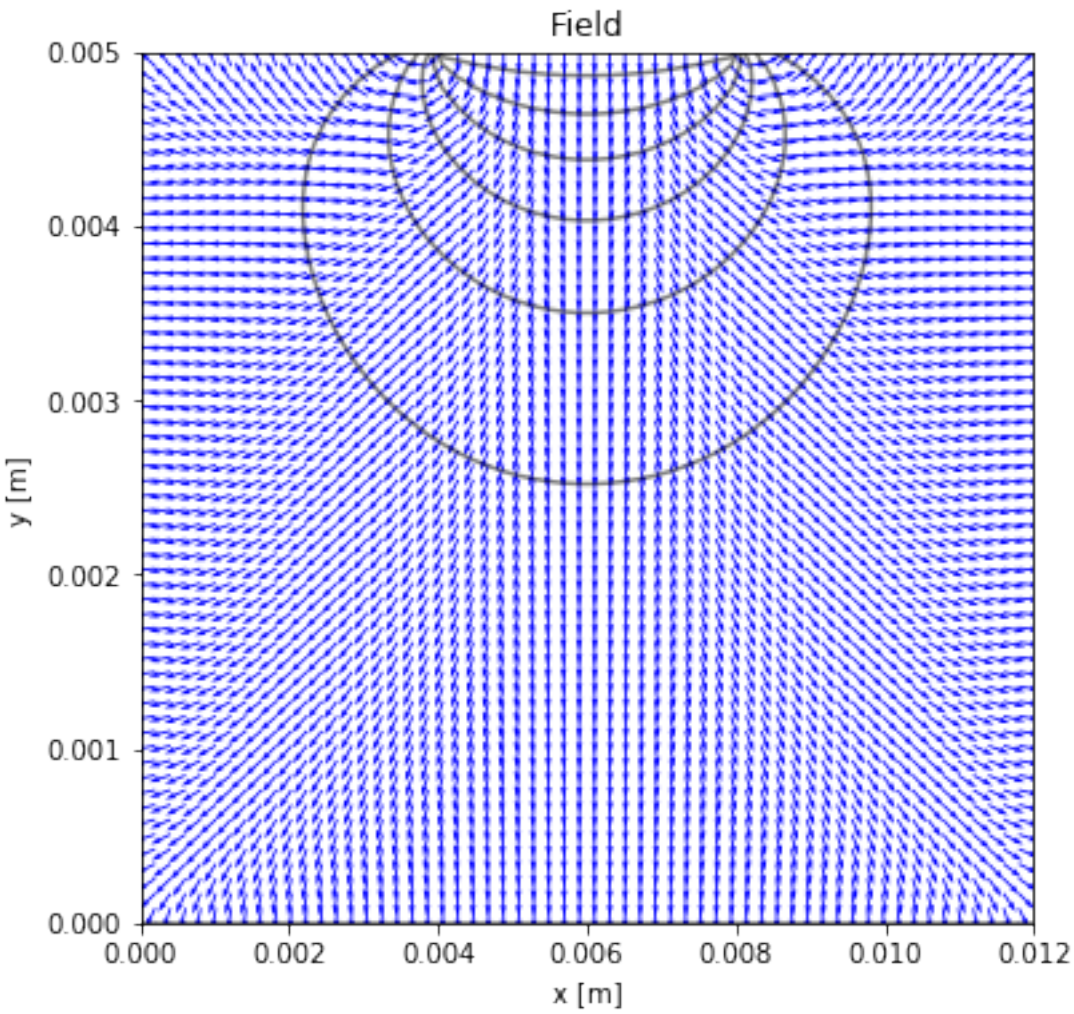
```
V, I, xx, yy = fd.init_problem_2(N, Lx, Ly, Vp=[10,0], plot_nodes=True)
```

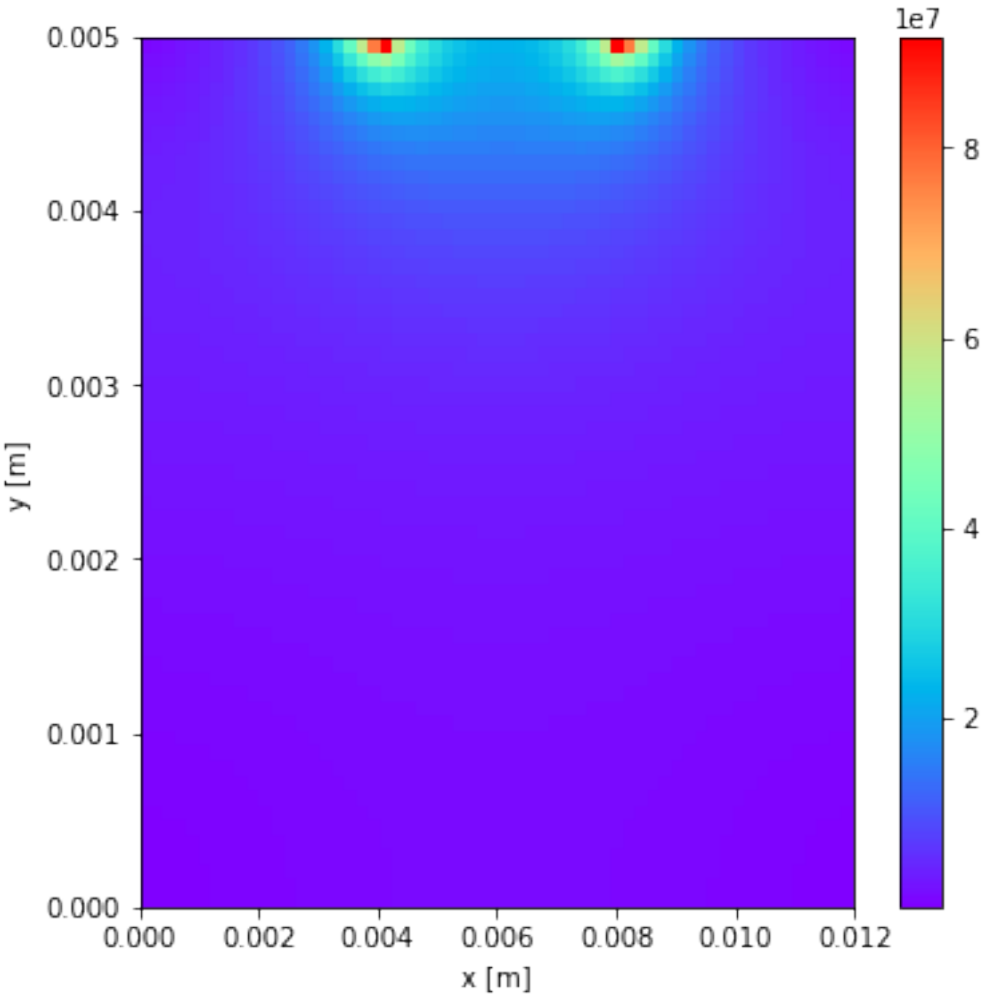


[12]:
```
V, Jx, Jy = fd.solve(N, V, I, matprop=3.7e7)
fd.plot_potential(V,xx,yy,12,"Potential")
fd.plot_field(Jx, Jy, V, xx, yy, "Field")
```


Potential
0.005
0.004
0.003
0.002
0.001
0.000
y [m]
0.000
0.002
0.004
0.006
0.008
0.010
0.012
x [m]
8
6
4
2
0


Potential
10
8
6
4
2
0
V
0.000
0.002
0.004
0.006
0.008
0.010
0.012
x [m]
0.000
0.001
0.002
0.003
0.004
0.005
y [m]
8
6
4
2


Field
0.005
0.004
0.003
0.002
0.001
0.000
y [m]
0.000
0.002
0.004
0.006
0.008
0.010
0.012
x [m]


1e7
8
6
4
2
0.005
0.004
0.003
0.002
0.001
0.000
y [m]
0.000
0.002
0.004
0.006
0.008
0.010
0.012
x [m]