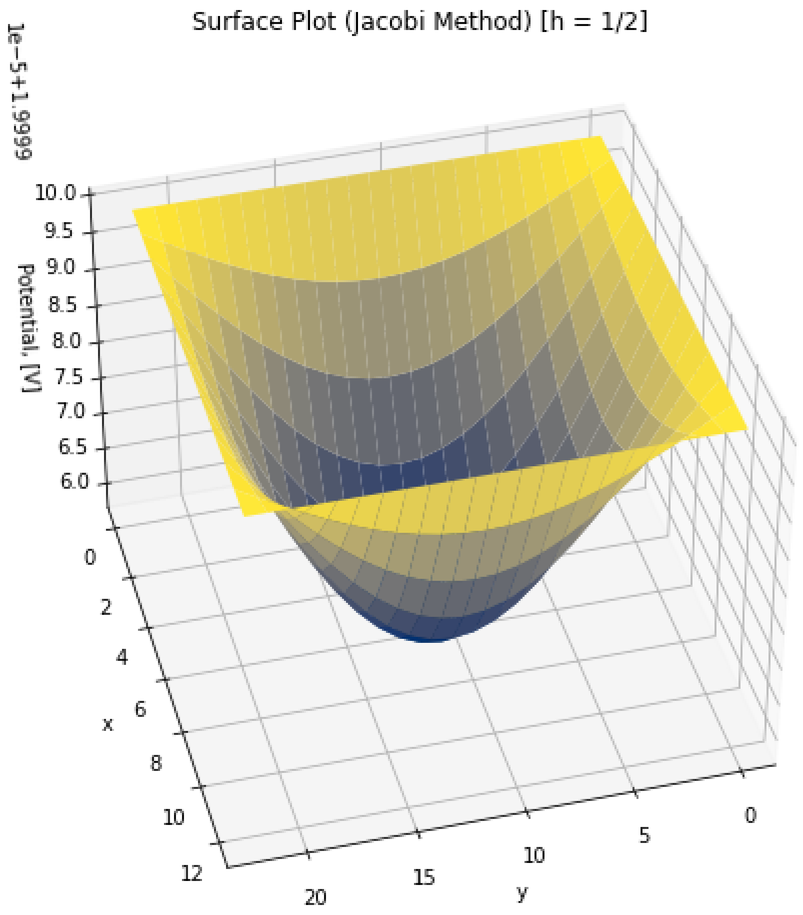
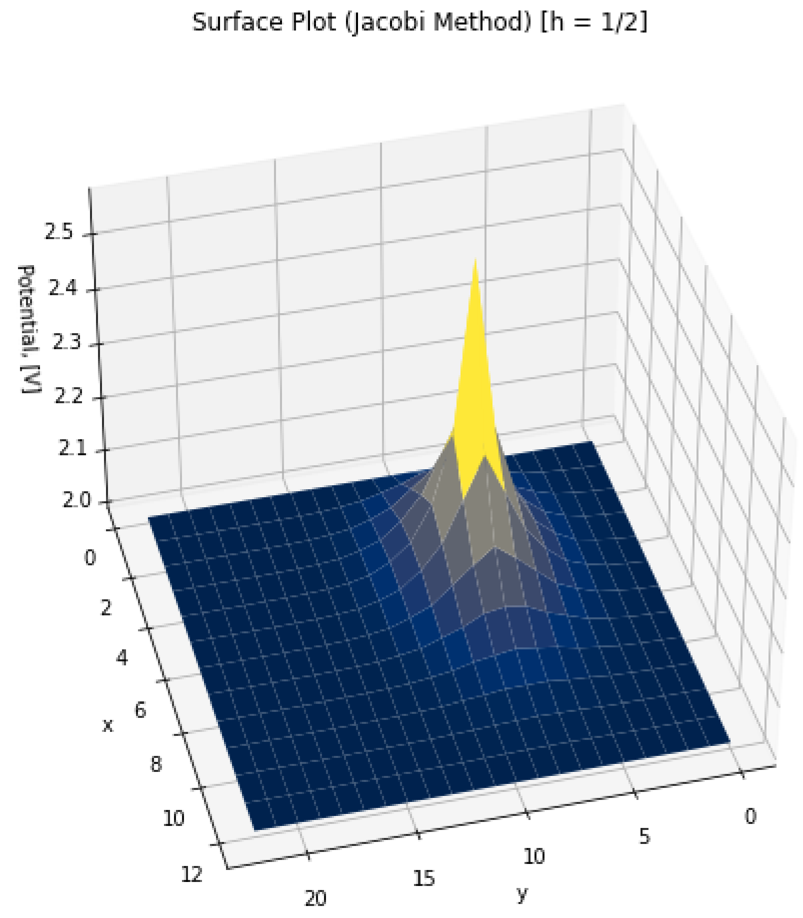
Jacobi's Method

Initially, neglecting the charge density it was calculated that 503 iterations were required to achieve the precision of 10^{-4}. later on at the interior sites, all the other charges were set to zero but the central charge was fixed to 5 A. The second graph with the concave nature is the one with the consideration of charge density in the calculation.

In [25]:
```
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt

#Setting up the boundary in the dimensions Lx = 11 and Ly = 21


h = 1/2
rectangle = (13, 23)




def SetBoundary(V):
    for x in range(23):
        V[0][x] = 2
        V[12][x] = 2
    for x in range(12):
        V[x][0] = 2
        V[x][22] = 2




def CheckPrecision(V1, V0):
    S = 0
    for i in range(1,12):
        for j in range(1, 22):
            S += V1[i][j] - V0[i][j]
    if S < 10**(-4):
        return False
    else:
        return True




 #Creating Charge Matrix

def Charge(Q):
    for x in range(8, 9):
        for y in range(4,5):
            Q[y][x] = 5

Q = np.zeros(rectangle)
Qm = np.zeros(rectangle)
Charge(Qm)







V = np.zeros(rectangle)
```

```
SetBoundary(V)

V1 = np.zeros(rectangle)
SetBoundary(V1)

#Jacobi Approach:
def Jacobi(V, h, Qm):
    k = 1


    status = True

    while(status):
        for i in range(1,12):
            for j in range(1,22):
                temp = V[i+1][j] + V[i-1][j]+ V[i][j+1]+V[i][j-1] + h**2*Qm[i
                V1[i][j] = temp/4
        for i in range(1, 12):
            V[i][11] = V[i][10]
        status = CheckPrecision(V1, V)
        k += 1
        for i in range(1,12,1):
            for j in range(1,22,1):
                V[i][j] = V1[i][j]






    x = np.arange(0,13, 1)
    y = np.arange(0,23,1)


    X,Y = np.meshgrid(y,x)



    fig, ax = plt.subplots(1, 1, figsize =(10, 8), subplot_kw ={'projection':
    ax.set_xlabel('y')
    ax.set_ylabel('x')
    ax.set_zlabel('Potential, [V]')
    ax.view_init(elev=40, azim =75)
    ax.set_title('Surface Plot (Jacobi Method) [h = 1/2]', y =1.1)
    ax.text(20,1, 10, 'Number of iterations = {}'.format(k))
    ax.plot_surface(X, Y, V1, cmap='cividis')


Jacobi(V, h, Q)

Jacobi(V, h, Qm)
```



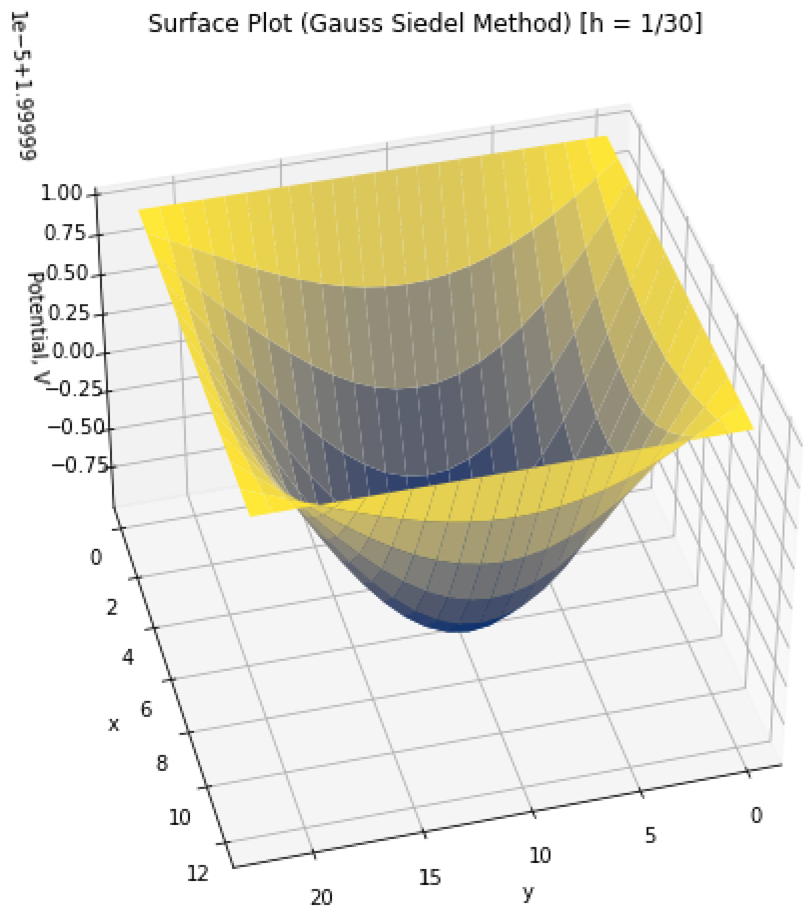


Gauß-Seidel Model

Using the Gauß-Siedel Model, the iteration converged a bit faster than that of the Jacobi's Model. In this model, 270 iterations were required for the solution to converge. Considering the charge densities in this model, furtheremore, reduced the iteration number which was 89.

In [15]:
```
#Gauss-Seidel Method

V = np.zeros(rectangle)
SetBoundary(V)

V1 = np.zeros(rectangle)
SetBoundary(V1)

k = 1


status = True

while(status):
    for i in range(1,12):
        for j in range(1,22):
            temp = V[i +1][j] + V1[i-1][j]+ V[i][j+1]+V1[i][j-1]
            V1[i][j] = temp/4
    for i in range(1, 12):
        V[i][11] = V[i][10]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1,12,1):
        for j in range(1,22,1):
            V[i][j] = V1[i][j]


print('Number of required iterations',k)



x = np.arange(0,13, 1)
y = np.arange(0,23,1)


X,Y = np.meshgrid(y,x)


fig, ax = plt.subplots(1, 1, figsize =(10, 8), subplot_kw ={'projection':'3d'
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim =75)
ax.set_title('Surface Plot (Gauss Siedel Method) [h = 1/30]', y =1.1)
ax.text(-1,20, 1.1, 'Number of iterations = {}'.format(k))
ax.plot_surface(X, Y, V1, cmap='cividis')
```

```
Number of required iterations 270
```



Successive over relaxation (SOR) Method

In this model we needed to calculate the parameter ω as:

$$\omega = \frac{2}{1 + \frac{\pi}{L}} \tag{1}$$

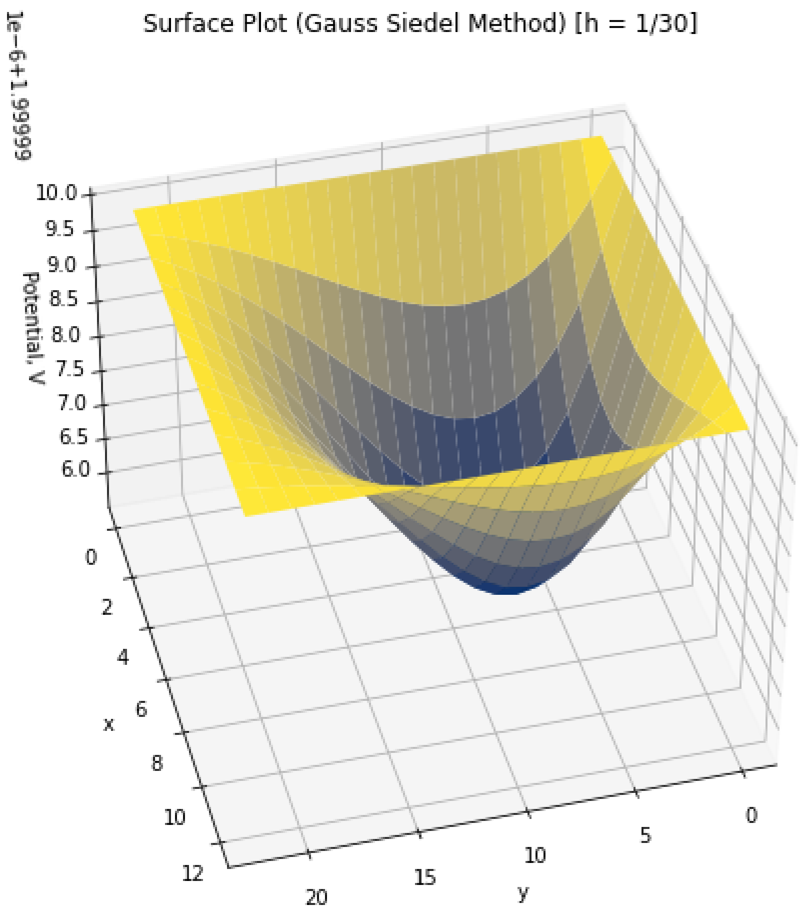
So value of L was taken to be 11 m for convinience. This result in the value of ω to be 0< (ω = 1.56) <2, which is the validation for the convergence of solution using this mode. And the solution converged faster with only 78 iterations.

With the consideration of the charge density, the iteration number required was lowered to 34.

In [22]:
```
#Successive Overrelaxation

w = 2/(1 + np.pi/11)
print('Omega =',w)

V = np.zeros(rectangle)
SetBoundary(V)

V1 = np.zeros(rectangle)
SetBoundary(V1)

ks = 1

status = True
```

```
while(status):
    for i in range(1,12,1):
        for j in range(1,22,1):
            temp = V[i +1][j] + V1[i-1][j]+ V[i][j+1]+V1[i][j-1]
            V1[i][j] = temp*w/4 + (1-w)*V[i][j]
    for i in range(1, 12):
        V[i][11] = V[i][10]
    status = CheckPrecision(V1, V)
    ks += 1
    for i in range(1,12,1):
        for j in range(1,22,1):
            V[i][j] = V1[i][j]


print('Number of required iterations =',ks)



x = np.arange(0,13, 1)
y = np.arange(0,23,1)


X,Y = np.meshgrid(y,x)


fig, ax = plt.subplots(1, 1, figsize =(10, 8), subplot_kw ={'projection':'3d'
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim =75)
ax.set_title('Surface Plot (Gauss Siedel Method) [h = 1/30]', y =1.1)
ax.text(1.0, 0, 120, 'Number of iterations = {}'.format(k))
ax.plot_surface(X, Y, V1, cmap='cividis')
plt.savefig('J30')
```

```
Omega = 1.5556946476191547
Number of required iterations = 78
```



Overrall validation

The 3d diagrams shown above which are in tred of converging to a point provides a overview that our program was in the right path.In general it was observed that all the plots agreed on the given set of boundary conditions. Futhermore, the convergent rate of SOR model was faster with lower iteration step of 78 and the convergent rate of Jacobs model was slower with 503 iteration steps also provides us the concrete proof for the correctness of the program executed. With the consideration of the charge densitiy (Second configuration) and a appropriate step size, the iterative step was somehow lower for all the methods as expected.