1 Question 1

Following Numerical Recipes, we will do a Von Neumann stability analysis on the leapfrog scheme for the advection equation:

$$\frac{f(t+dt,x)-f(t-dt,x)}{2dt} = -v\frac{f(t,x+dx)-f(t,x-dx)}{2dx}$$
$$f(t+dt,x) = f(t-dt,x) - a\left[f(t,x+dx)-f(t,x-dx)\right]$$

Substituting a complex exponential solution of the form: $f(x,t)=\xi^t \exp(ikx)$ gives:

$$\xi^{t+dt}e^{ikx} = \xi^{t-dt}e^{ikx} - a\,\xi^t e^{ikx}(e^{ikdx}-e^{-ikdx})$$

Dividing both sides by $\xi^t e^{ikx}$:

$$\xi^{dt} = \xi^{-dt} - a(e^{ikdx}-e^{-ikdx})$$
$$\xi^{dt} = \xi^{-dt} - 2ia\sin(kdx)$$

Multiplying by ξ^{dt} and rearranging:

$$(\xi^{dt})^2 + 2\,\xi^{dt} ia\sin(kdx) - 1 = 0$$

$$\Rightarrow \xi^{dt} = \frac{-2ia\sin(kdx) \pm \sqrt{(2ia\sin(kdx))^2+4}}{2} = -ia\sin(kdx) \pm \sqrt{1-a^2\sin^2(kdx)}$$

If $a > 1$:

$$\Rightarrow 1 - a^2\sin^2(kdx) < 0$$
$$\Rightarrow \xi^{dt} = -ia\sin(kdx) \pm i\sqrt{a^2\sin^2(kdx)-1} = i\left(a\sin(kdx) \pm \sqrt{a^2\sin^2(kdx)-1}\right)$$
$$|\xi^{dt}|^2 = \left(a\sin(kdx) \pm \sqrt{a^2\sin^2(kdx)-1}\right)^2$$
$$= a^2\sin^2(kdx) + 1 - a^2\sin^2(kdx) \pm a\sin(kdx)\sqrt{a^2\sin^2(kdx)-1}$$
$$= 1 \pm a\sin(kdx)\sqrt{a^2\sin^2(kdx)-1} = \left(|\xi(k)|^2\right)^{dt} \neq 1$$
$$\Rightarrow |\xi(k)|^2 \neq 1$$

If $a \leq 1$:

$$\Rightarrow 1 - a^2\sin^2(kdx) \geq 0$$
$$\Rightarrow \xi^{dt} = -ia\sin(kdx) \pm \sqrt{1-a^2\sin^2(kdx)}$$
$$|\xi^{dt}|^2 = a^2\sin^2(kdx) + 1 - a^2\sin^2(kdx) = 1$$
$$\Rightarrow |\xi(k)|^2 = 1$$

Since energy is proportional to $|\xi^t|^2$ (the amplitude squared of the assumed solution), energy is conserved only when $|\xi|^2 = 1$, since $1^t = 1$ for all t. Therefore energy is conserved only when the CFL condition is met: $a = v\frac{dt}{dx} \leq 1$.

2 Question 2

a)

For the relaxation method, each iteration replaces the updates the new potential as the charge density plus the average of its neighbour's old potentials. If we set $V[0,0] = \rho[0,0] = 1$, then:

$$V[0,0] = \rho[0,0] + \frac{1}{4}(V[1,0] + V[-1,0] + V[0,1] + V[0,-1])$$

$$\Rightarrow V[1,0] + V[-1,0] + V[0,1] + V[0,-1] = 0$$

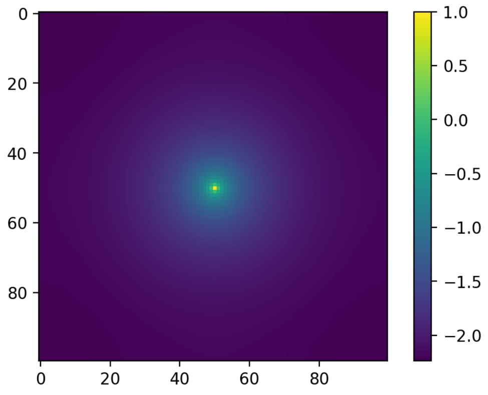
But $V[1,0] = V[-1,0] = V[0,1] = V[0,-1]$:

$$\Rightarrow V[1,0] = 0$$

Since the potential at $V[1,0]$ lacks the radial symmetry like $V[0,0]$ to solve for it exactly, we will need to use a program to update the potential in iterations. Here is the code used to do so:

```
n=100
V=np.zeros([n,n])
rho=np.zeros([n,n])
rho[0,0]=1
niter=n*20

for i in tqdm(range(niter)):
    Vavg=1/4*(np.roll(V,1,0)+np.roll(V,-1,0)+np.roll(V,1,1)+np.roll(V,-1,1))
    Vnew=rho+Vavg
    V=Vnew
    V=V+(1-V.max())  # offset potential to keep V[0,0]=1
    if i%10==0:
        plt.clf()
        plt.imshow(np.fft.fftshift(V))  # fftshift to plot negative indices
        plt.colorbar()
        plt.pause(0.01)
```

After iterating, the final potential was plotted, and the final values for the potentials $V[1,0]$, $V[2,0]$, $V[3,0]$ were recorded:



After 2000 iterations:

$$V[1,0] = 0.0003268678913179862$$

$$V[2,0] = -0.45286722549833636$$

$$V[5,0] = -1.0473704532703243$$

b)

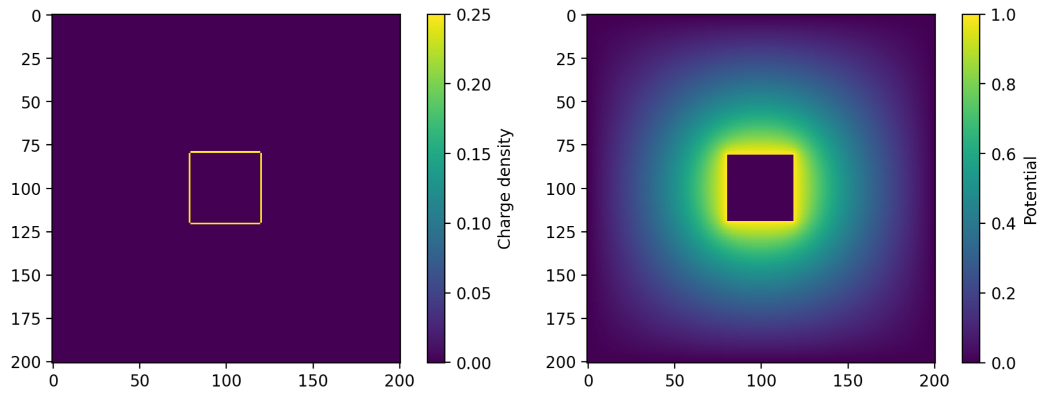
To find the charge density, the edges and inside of the box were masked, while setting the potential boundary conditions at the edges of the box to 1. The conjugate gradient method written in class was adopted to use a new matrix multiplication. Here are the functions written for this conjugate gradient method:

```
def avg_neighbors(mat):
    up=np.roll(mat,1,0)
    down=np.roll(mat,-1,0)
    right=np.roll(mat,1,1)
    left=np.roll(mat,-1,1)
    avg=1/4*(up+down+right+left)
    return avg

def mat_mult(x,mask):
    x[mask]=0  # We only care about interior, apply mask
    avg=avg_neighbors(x)
    avg[mask]=0  # Boundaries become non-zero after avg'ing
    return x-avg

def conjgrad(b,xinit,mask,niter,plot=False):  # CG method from class
    r=b-mat_mult(xinit,mask)
    p=r.copy()
    rr=np.sum(r**2)
    x=xinit
    for i in range(niter):
        Ap=mat_mult(p,mask)
        pAp=np.sum(p*Ap)  # Can't use @ for dot product when not vectors
        alpha=rr/pAp
        x=x+alpha*p
        r=r-alpha*Ap
        rr_new=np.sum(r**2) # Same for this dot product
        beta=rr_new/rr
        p=r+beta*p
        rr=rr_new
        b=mat_mult(x,mask) # Calculate and plot rho
        if plot:
            if i%5==0:
                plt.clf()
                plt.imshow(b)
                plt.colorbar(label='Charge density')
                plt.title('Iteration '+str(i))
                plt.pause(0.01)
    return x,b
```

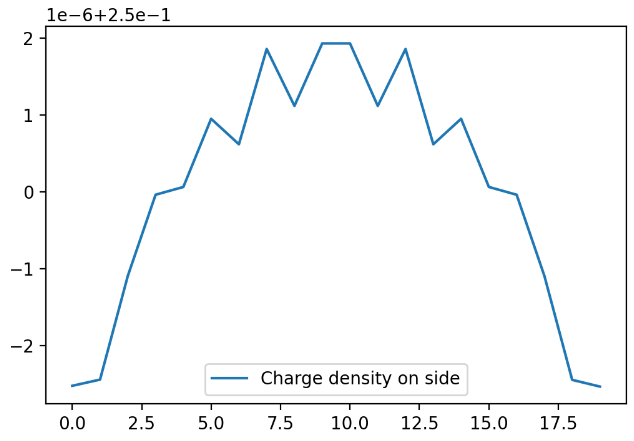
Here are the boundary conditions and masks being set:

```
n=101
mask=np.zeros([n,n],dtype='bool') # Mask for edges and inside box
mask[0,:]=True
mask[-1,:]=True
mask[:,0]=True
mask[:,-1]=True
mask[2*n//5:3*n//5,2*n//5:3*n//5]=True
bc=np.zeros([n,n])  # Potential bc's on edges of box
bc[2*n//5,2*n//5:3*n//5]=1
bc[3*n//5-1,2*n//5:3*n//5]=1
bc[2*n//5:3*n//5,2*n//5]=1
bc[2*n//5:3*n//5,3*n//5-1]=1

bc_rhs=avg_neighbors(bc)  # Move boundary conditions to RHS
bc_rhs[mask]=0
b=bc_rhs
V,rho=conjgrad(b,0*b,mask,niter=n)
V[mask]=bc[mask] # Add back BC's after zeroing with mask
```

The resulting charge density and potentials were plotted after iterating:



Plotting the charge density along one side of the box shows that it is a constant value of 0.25 with small deviations of order 10^6

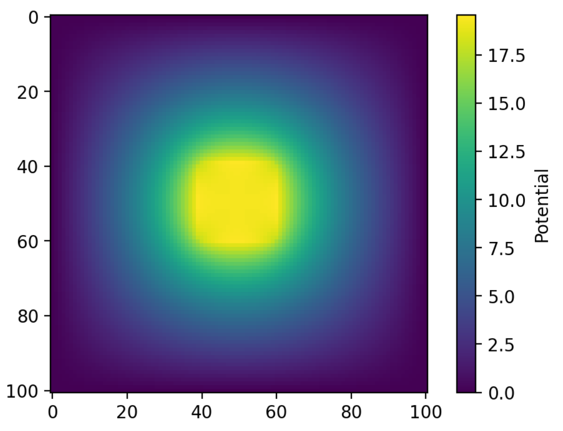


c)

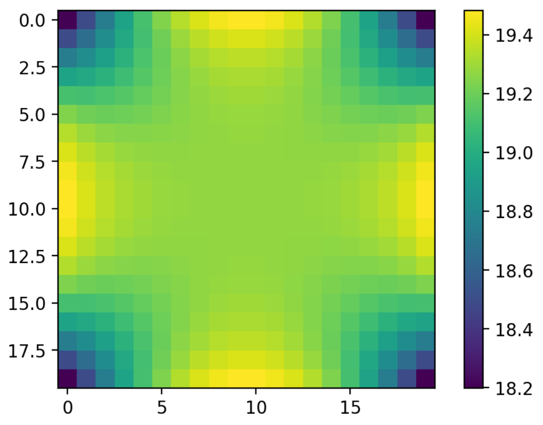
To get the potential everywhere, the same conjugate gradient method was used with b set as the charge density calculated in part b). This time, only the edges were uses as a mask and not the inside of the box:

```
mask=np.zeros([n,n],dtype='bool') # Mask just for edges
mask[0,:]=True
mask[-1,:]=True
mask[:,0]=True
mask[:,-1]=True

Vall=conjgrad(rho,0*rho,mask,niter=n)[0]
plt.imshow(Vall)
```

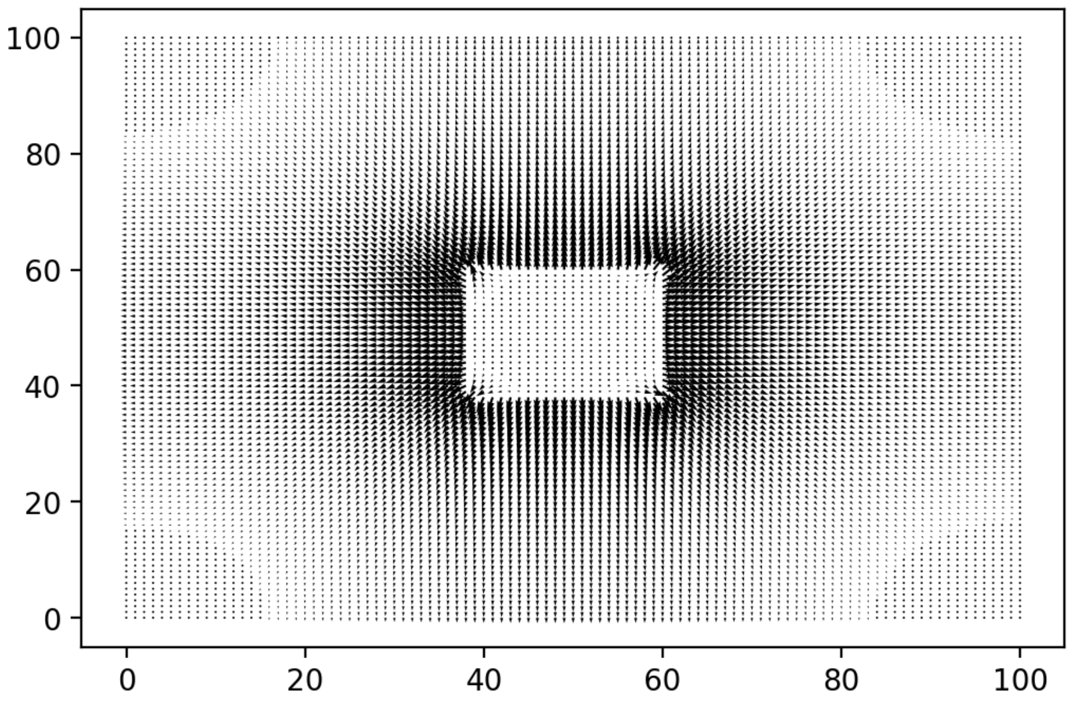
The plot of the potential everywhere is on the next page:



The potential can be seen to be quite constant in the plot above. Plotting the potential of just the inside of the box shows just how constant this is. There is a small variation along the edges, while a more constant value is reached in the center:



Finally, we can plot the electric field to determine if it is indeed perpendicular to the surface. Taking a the difference between successive x and y points of the potential to the be respective components of the electric field, `matplotlib.pyplot.quiver` was used to plot the vector field (next page):



The above plot shows the electric field to be perpidicular along the edges, and is stronger near the points of the cube.