Guillaume Payeur (260929164)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 200
plt.rcParams.update({"text.usetex": True})
from scipy.signal import convolve2d as conv2d
```

Q1

As suggested, we assume that $f(x,t)$ can be decomposed into it's eigenmodes, each of which are independent solutions to the PDE. $f(x,t)$ is then a sum of it's eigenmodes, each of which satisfy the PDE. Energy is therefore conserved for $f(x,t)$ if energy is conserved for each of it's eigenmodes. Therefore, we may consider WLOG $f(x,t)$ of the form

$$f(x,t)=\xi^t\exp(ikx) \tag{1}$$

ie, $f(x,t)$ is a single eigenmode. Energy is conserved if and only if $|\xi|=1$, since since otherwise $|f(x,t)|$ either grows or decreases with time, with limiting value $|f(x,t)|\to\infty|$ or $|f(x,t)|\to 0$ respectively. We therefore plug in the equation for $f(x,t)$ in

$$\frac{f(t+dt,x)-f(t-dt,x)}{2dt}=-v\frac{f(t,x+dx)-f(t,x-dx)}{2dx} \tag{2}$$

and expect to see $|\xi|=1$ assuming

$$\alpha\equiv\frac{vdt}{dx}\leq 1. \tag{3}$$

We get

$$\frac{\xi^{t+dt}e^{ikx}-\xi^{t-dt}e^{ikx}}{2dt}=-v\frac{\xi^te^{ik(x+dx)}-\xi^te^{ik(x-dx)}}{2dx} \tag{4}$$

$$\implies \xi^{t+dt}e^{ikx}-\xi^{t-dt}e^{ikx}=-\frac{2vdt}{2dx}\left(\xi^te^{ik(x+dx)}-\xi^te^{ik(x-dx)}\right) \tag{5}$$

$$\implies \xi^{t+dt}e^{ikx}-\xi^{t-dt}e^{ikx}=-\alpha\left(\xi^te^{ik(x+dx)}-\xi^te^{ik(x-dx)}\right) \tag{6}$$

Multiplying both sides by $\xi^{-t+dt}e^{-ikx}$, we get

$$\xi^{2dt}-1=\xi^{dt}\left(-\alpha\left(e^{ikdx}-e^{-ikdx}\right)\right) \tag{7}$$

Using Euler's identity $e^{ix}=\cos(x)+i\sin(x)$, we get

$$\implies \xi^{2dt} - 1 = \xi^{dt}\Big(- \alpha(2i\sin(kdx))\Big) \tag{8}$$
$$\implies (\xi^{dt})^2 + (2i\alpha\sin(kdx))\xi^{dt} - 1 = 0 \tag{9}$$

This is a quadratic equation in ξ^{dt}. Using the Quadratic formula, we get

$$\xi^{dt} = \frac{-2i\alpha\sin(kdx) \pm \sqrt{-4\alpha^2\sin^2(kdx)+4}}{2} \tag{10}$$
$$= -i\alpha\sin(kdx) \pm \sqrt{-\alpha^2\sin^2(kdx)+1} \tag{11}$$

Now, assuming $\alpha < 1$, and using that $|\sin(x)| < 1\ \forall x$ and $|ab| = |a||b|$, we have that

$$|\alpha^2\sin^2(kdx)| < 1 \tag{12}$$

Moreover, α^2 and $\sin^2(kdx)$ are real and positive. Therefore, $-\alpha^2\sin^2(kdx)+1$ is real and $\forall x$,

$$-\alpha^2\sin^2(kdx)+1 > 0 \tag{13}$$

It follows that the real and imaginary parts of ξ^{dt} are

$$\Re(\xi^{dt}) = \pm\sqrt{-\alpha^2\sin^2(kdx)+1} \tag{14}$$
$$\Im(\xi^{dt}) = -\alpha\sin(kdx) \tag{15}$$

Therefore, using the fact that for a complex number z, $|z| = (\Re(z))^2 + (\Im(z))^2$, we get

$$|\xi^{dt}| = (-\alpha\sin(kdx))^2 + \Big(\pm\sqrt{-\alpha^2\sin^2(kdx)+1}\Big)^2 \tag{16}$$
$$= \alpha^2\sin^2(kdx) + (-\alpha^2\sin^2(kdx)+1) \tag{17}$$
$$= 1 \tag{18}$$

Since $dt \neq 0$, this means

$$|\xi| = 1 \tag{19}$$

which we argued above guarantees conservation of energy for any $f(x,t)$. Since $\alpha < 1$ was chosen arbitrarily it follows that energy is conserved if the CFL condition is satisfied.

Q2

a)

We adapt the code we wrote in class so that we can get the potential from a fixed charge density distribution.

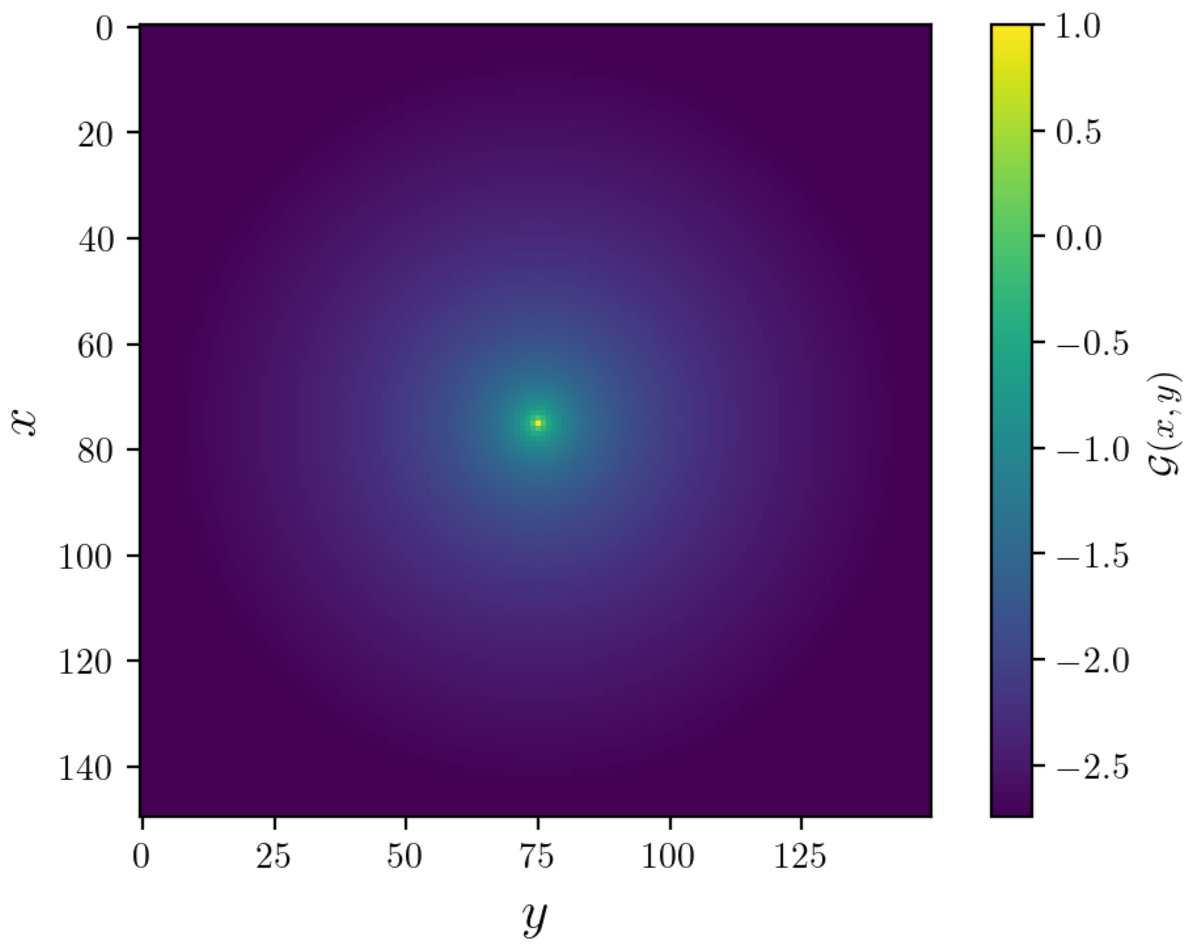
In [2]:
```python
def average_neighbors(mat):
    out=0*mat
    out=out+np.roll(mat,1,0)
    out=out+np.roll(mat,-1,0)
    out=out+np.roll(mat,1,1)
    out=out+np.roll(mat,-1,1)
    return out/4

class Grid:
    def __init__(self,bc,mask):
        self.bc=bc
        self.mask=mask
    def make_rhs(self):
        rhs=self.bc
        return rhs
    def __matmul__(self,x):
        ave=average_neighbors(x)
        return (x-ave)/(dx**2)

def conjgrad(A,b,x=None,niter=100,plot=False):
    if x is None:
        x=0*b
    r=b-A@x
    p=r.copy()
    rtr=np.sum(r**2)
    for i in range(niter):
        Ap=A@p
        #pAp=p@Ap #wrong if p,Ap aren't already vectors!
        pAp=np.sum(p*Ap)
        alpha=rtr/pAp
        x=x+alpha*p
        r=r-alpha*Ap
        rtr_new=np.sum(r**2)
        beta=rtr_new/rtr
        p=r+beta*p
        rtr=rtr_new
    return x
```

From the functions defined above I generate the Green's function for the Laplacian, valid in a circle of radius roughly 75

In [3]:
```
n=150
dx=1
mask=np.zeros([n,n],dtype='bool')
bc=np.zeros([n,n])

bc[n//2,n//2] = 1/(dx**2)
mask[n//2,n//2] = True

A=Grid(bc,mask)
b=A.make_rhs()

x=conjgrad(A,b,niter=100)
Greens=x.copy()
```

I plot the resulting Green's function potential

In [4]:
```
plt.imshow(Greens-Greens[n//2,n//2]+1)
plt.colorbar(label='$\mathcal{G}(x,y)$')
plt.xlabel('$y$',fontsize=15)
plt.ylabel('$x$',fontsize=15)
plt.show()
```



The potential at (5,0) is -1.05 as shown here

In [5]:
```
(Greens-Greens[n//2,n//2]+1)[n//2+5,n//2]
```

Out[5]:
```
-1.0516083785889894
```

And we get that the potential at (1,0) and (2,0) are

In [6]:
```
print((Greens-Greens[n//2,n//2]+1)[n//2+1,n//2])
print((Greens-Greens[n//2,n//2]+1)[n//2+2,n//2])
```

```
1.3322676295501878e-15
-0.45367357278235554
```

$$V(1,0) = 0 \tag{20}$$
$$V(2,0) = -0.45 \tag{21}$$

b)

We first write a solver to find the charge on a square box held at a potential of 1. This is pretty much all code we wrote in class.

In [7]:
```
def average_neighbors(mat):
    out=0*mat
    out=out+np.roll(mat,1,0)
    out=out+np.roll(mat,-1,0)
    out=out+np.roll(mat,1,1)
    out=out+np.roll(mat,-1,1)
    return out/4

class Grid:
    def __init__(self,bc,mask):
        self.bc=bc
        self.mask=mask
    def make_rhs(self):
        rhs=average_neighbors(self.bc)
        rhs[self.mask]=0 #we need to zero out the RHS on the boundary
        return rhs
    def __matmul__(self,x):
        x[self.mask]=0
        ave=average_neighbors(x)
        ave[self.mask]=0
        return x-ave

def conjgrad(A,b,x=None,niter=100,plot=False):
    if x is None:
        x=0*b
    r=b-A@x
    p=r.copy()
    rtr=np.sum(r**2)
    for i in range(niter):
        Ap=A@p
        #pAp=p@Ap #wrong if p,Ap aren't already vectors!
        pAp=np.sum(p*Ap)
        alpha=rtr/pAp
        x=x+alpha*p
        r=r-alpha*Ap
        rtr_new=np.sum(r**2)
        beta=rtr_new/rtr
        p=r+beta*p
        rtr=rtr_new
    return x
```

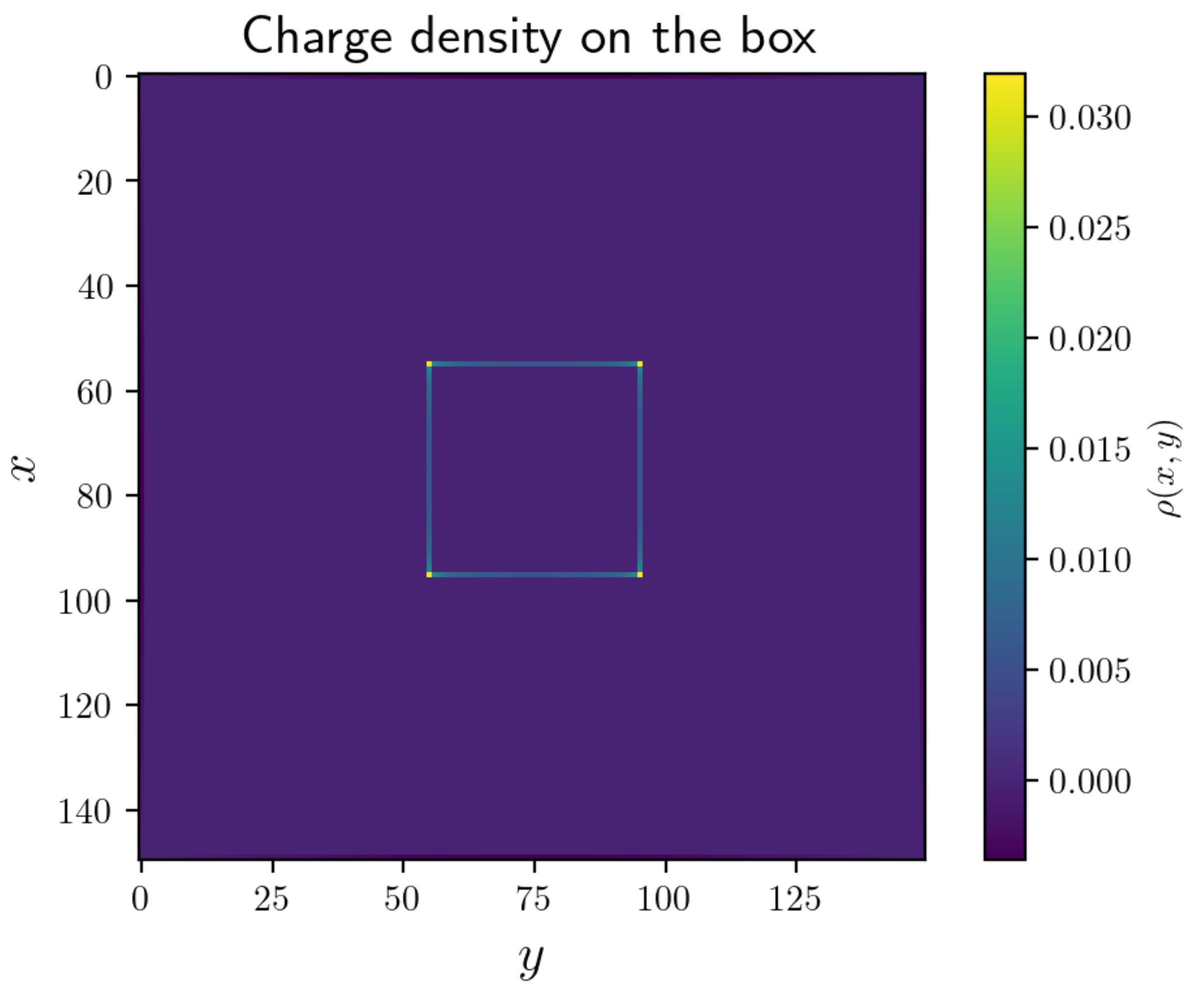
Using the solver to get the charge density on a square with sides of length 41

In [8]:
```
n=150
mask=np.zeros([n,n],dtype='bool')
bc=np.zeros([n,n])
x=np.linspace(-1,1,n)
xsqr=np.outer(x**2,np.ones(n))
rsqr=xsqr+4*xsqr.T
mask[:,0]=True
mask[0,:]=True
mask[-1,:]=True
mask[:,-1]=True

bc[n//2+20,n//2-20:n//2+21]=1.0
bc[n//2-20,n//2-20:n//2+21]=1.0
bc[n//2-20:n//2+21,n//2-20]=1.0
bc[n//2-20:n//2+21,n//2+20]=1.0
mask[n//2+20,n//2-20:n//2+21]=True
mask[n//2-20,n//2-20:n//2+21]=True
```

```
mask[n//2-20:n//2+21,n//2-20]=True
mask[n//2-20:n//2+21,n//2+20]=True

A=Grid(bc,mask)
b=A.make_rhs()

x=conjgrad(A,b,niter=3*n)
V=x.copy()
V[A.mask]=A.bc[A.mask]
Ex=V-np.roll(V,-1,0)
Ey=V-np.roll(V,-1,1)

rho=V-average_neighbors(V)
```

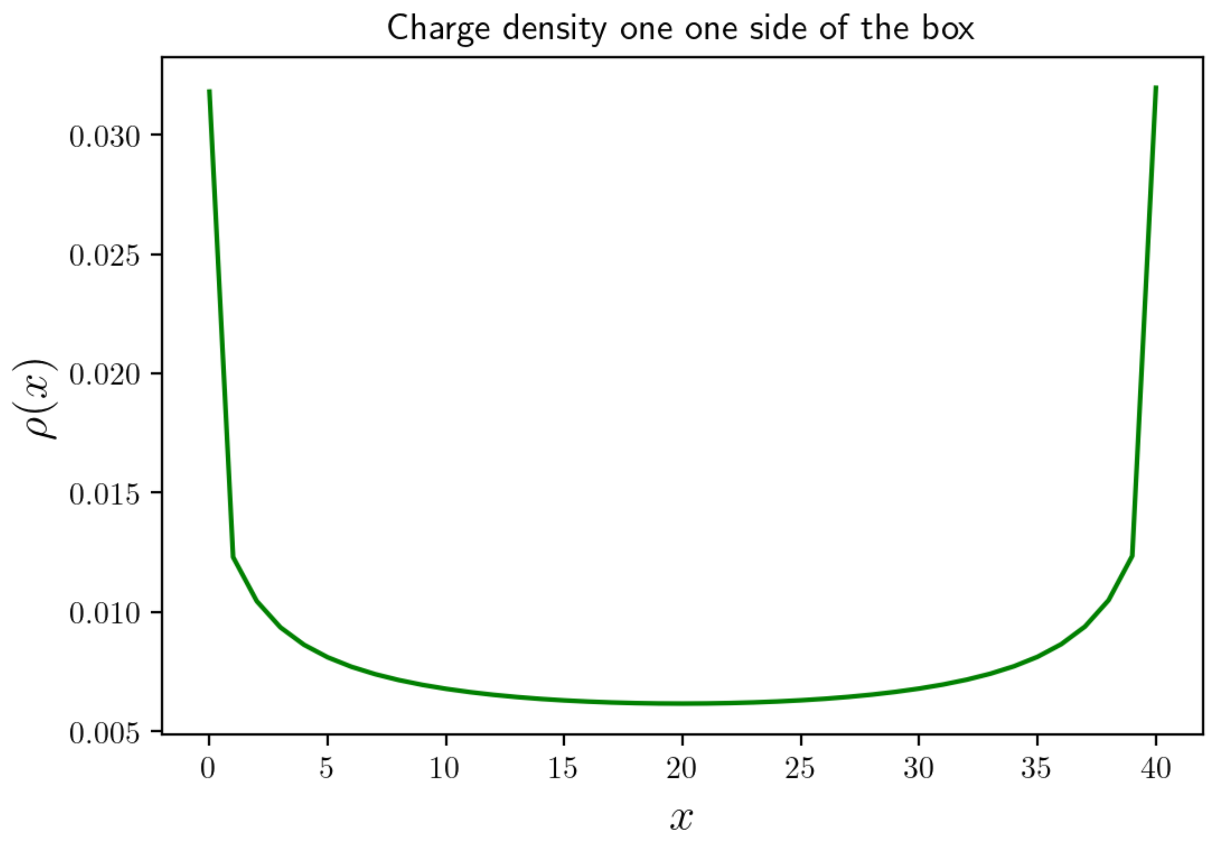
Plotting the charge density on the box

In [9]:
```
plt.imshow(rho)
plt.colorbar(label='$\\rho(x,y)$')
plt.xlabel('$y$',fontsize=15)
plt.ylabel('$x$',fontsize=15)
plt.title('Charge density on the box',fontsize=15)
plt.show()
```



Plotting the charge density specifically on one side of the box

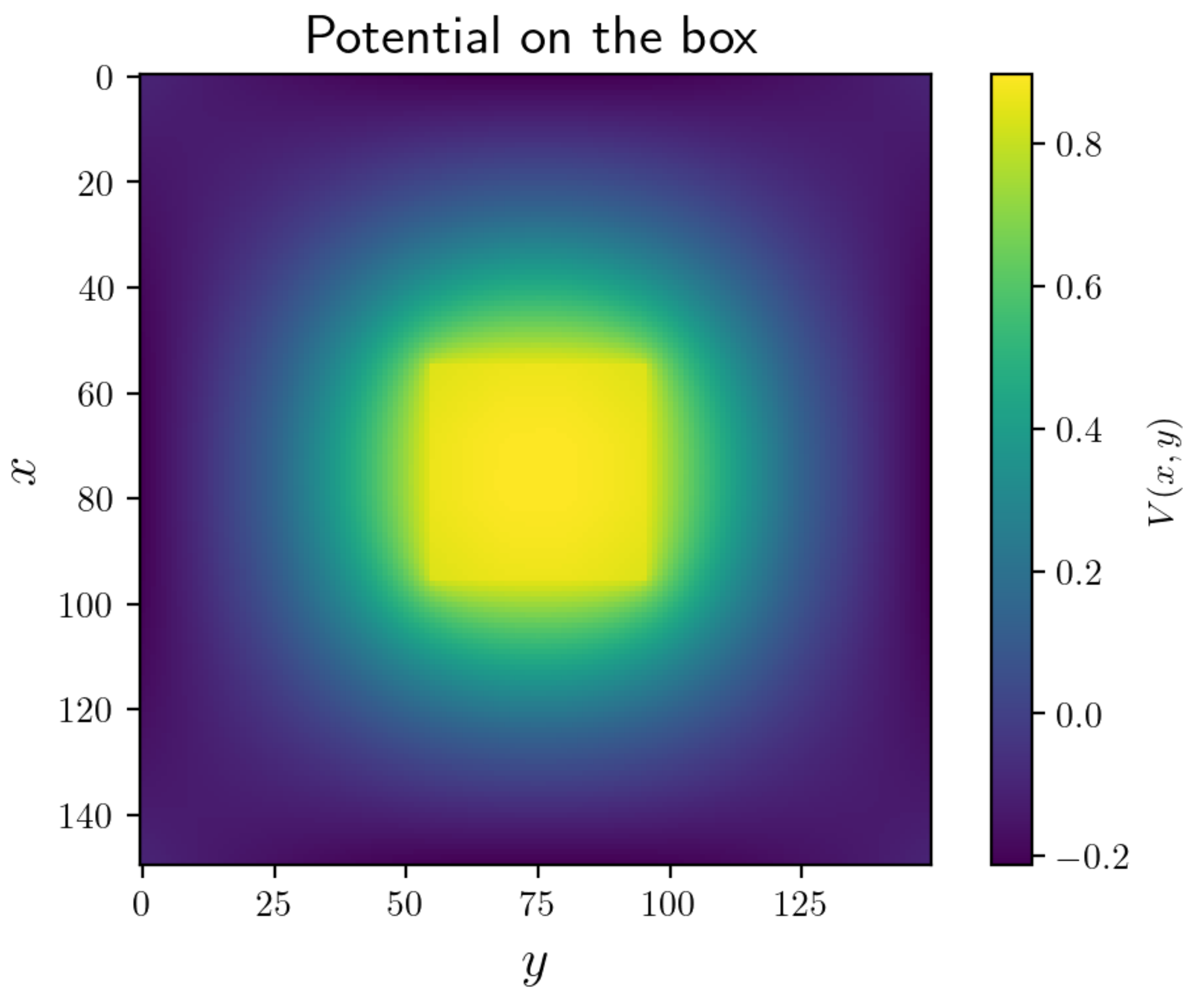
In [10]:
```
plt.plot(rho[95,55:96],color='green')
plt.title('Charge density one one side of the box')
plt.ylabel('$\\rho(x)$',fontsize=15)
plt.xlabel('$x$',fontsize=15)
plt.show()
```



c)

Now we convolve the Green's function computed earlier and the charge density found just above, to get the potential.

In [11]:
```
V = conv2d(Greens,rho)[150-75:150+75,150-75:150+75]
plt.imshow(V)
plt.colorbar(label='$V(x,y)$')
plt.xlabel('$y$',fontsize=15)
plt.ylabel('$x$',fontsize=15)
plt.title('Potential on the box',fontsize=15)
plt.show()
```



Let's see how close to constant the potential is inside the box, by finding the standard deviation and mean of the potential inside the box

In [12]:
```
std = np.std(V[75-20:75+21,75-20:75+21])
mean = np.mean(V[75-20:75+21,75-20:75+21])
print(std)
print(mean)
print(std/mean)
```

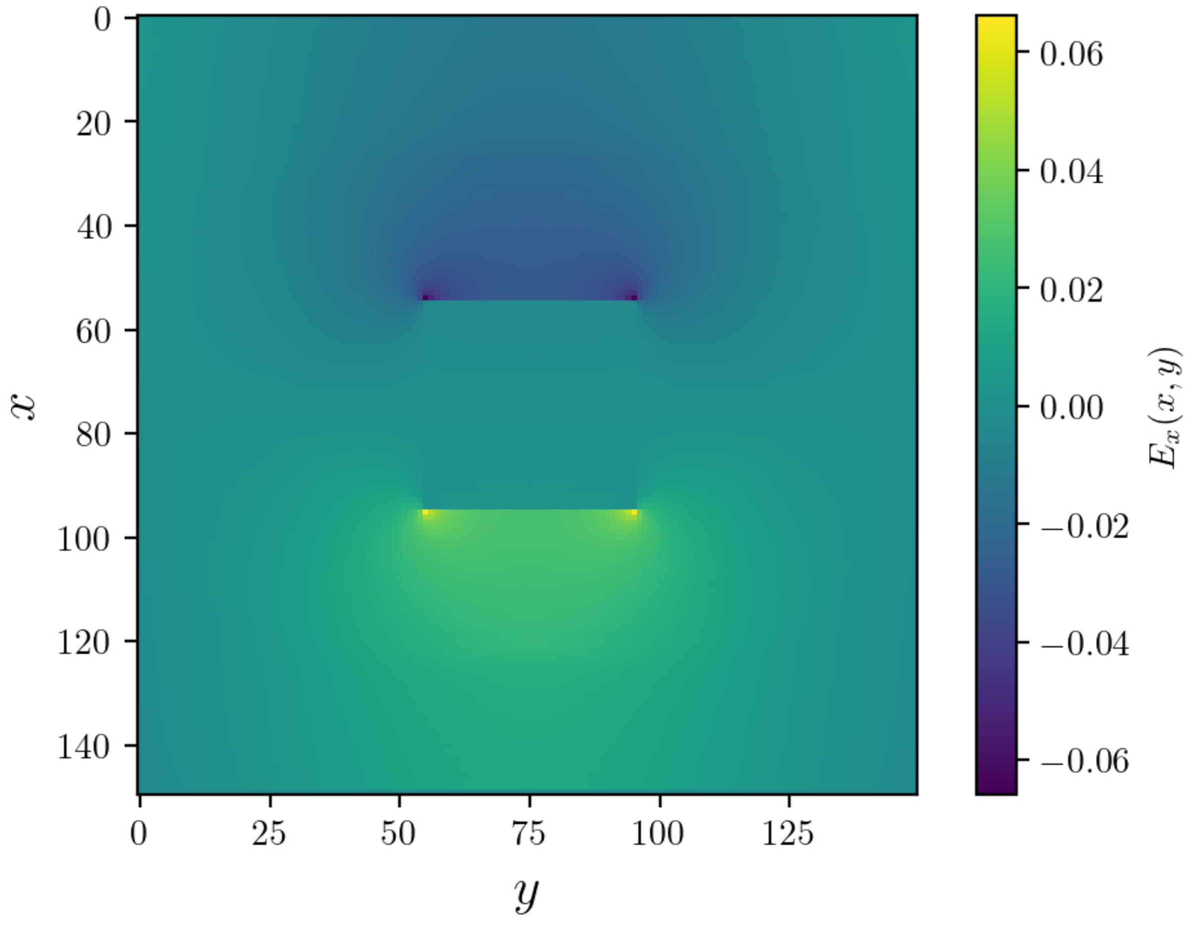
```
0.013388121877581914
0.8765030015969425
0.015274473508007912
```

So the variation in V inside the box is about 1% of the mean value of V, which is not much. So the potential is quite constant

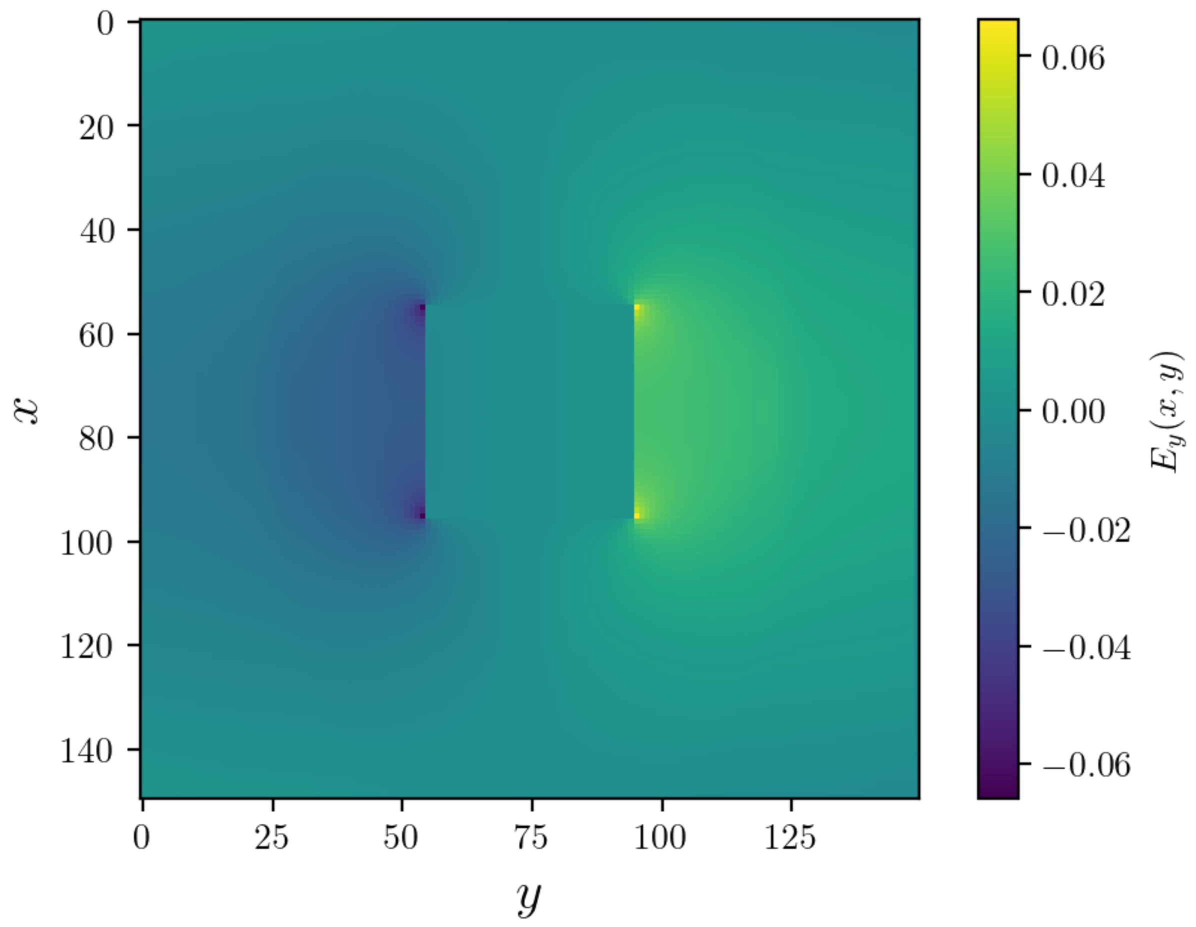
Now we plot the electric field

In [13]:
```
Ex=V-np.roll(V,-1,0)
Ey=V-np.roll(V,-1,1)
```

In [14]:
```
plt.imshow(Ex)
plt.colorbar(label='$E_x(x,y)$')
plt.xlabel('$y$',fontsize=15)
plt.ylabel('$x$',fontsize=15)
plt.show()
```



In [15]:
```
plt.imshow(Ey)
plt.colorbar(label='$E_y(x,y)$')
plt.xlabel('$y$',fontsize=15)
plt.ylabel('$x$',fontsize=15)
plt.show()
```



This matches everything we expected. In particular, E is perpendicular to the surface of the square, and it peaks at the corners of the square. It is also roughly null inside the square.