Homework 6: Computational Physics

Pablo Lopez Duque

November 4th, 2020

1 Problem 1

Neutron emitters, described by, $\dot{\rho} = D\rho'' + C\rho$ and absorbing boundary conditions $\rho(-L/2, t) = 0 = \rho(L/2, t)$, have a critical mass.

(a) **Show analytically that if reflecting boundary conditions are used that the neutron density will always diverge for large time, meaning there is no critical mass.**

We can start by solving the original differential equation using separation of variables:

$$\frac{\partial \rho}{\partial t} = D\frac{\partial^2 \rho}{\partial x^2} + C\rho \tag{1}$$

It is clear form the equation that the density is expected to increase over time if $C > 0$. If $C = 0$ we recover the diffusion equation which describes a system in which there is a natural transfer of particles from high density regions to low density ones until the density becomes uniform. Thus, when the spatial derivative ceases to change, the time derivative does so as well. The second term introduces a further complication into the analysis, so we must proceed with caution. Let's analyze it mathematically and interpret it only afterwards:

Absorbing boundary conditions

Let $\rho = \chi(x)T(t)$,

$$\Rightarrow \dot{\rho} = \chi(x)\dot{T}(t),$$
$$\rho'' = \chi''(x)T(t)$$
$$\Rightarrow \chi(x)\dot{T}(t) = D\chi''(x)T(t) + C\chi(x)T(t)$$

Now, we can divide both sides by $D\chi(x)T(t)$:

$$\Leftrightarrow \frac{\dot{T}(t)}{DT(t)} = \frac{\chi''(x)}{\chi(x)} + \frac{C}{D}$$

The two sides of the last equation must be equal. Hence they must be a constant. Let's call it κ. Consider the case when $\kappa > 0$ since we want to have a divergent solution in time in order to determine the critical conditions.

$$\Rightarrow \frac{\dot{T}(t)}{DT(t)} = \kappa$$
$$\frac{\chi''(x)}{\chi(x)} + \frac{C}{D} = \kappa$$

Hence, the solutions are:

$$T(t) = \exp(\kappa D t)$$

$$\chi(x) = A\sin\left(\sqrt{\frac{C}{D} - \kappa x}\right) + B\cos\left(\sqrt{\frac{C}{D} - \kappa x}\right)$$

At this point, in class we applied the absorbing boundary conditions (BC), which yield:

$$\chi(x) = A\sin\left(\sqrt{\frac{C}{D} - \kappa x}\right)$$

and constrained the value for κ, as follows:

$$\left(\sqrt{\frac{C}{D}-\kappa}\right)L = n\pi \Rightarrow \kappa = \frac{C}{D} - \left(\frac{n\pi}{L}\right)^2$$

Hence, κ can be either positive or negative depending on the value for n. Since we started with the assumption of an exponential function growing over time(i.e. a chain reaction), we want $\kappa > 0$. Also, the maximum attainable κ happens when $n = 1$. Hence, criticality is reached when $\frac{C}{D} > \left(\frac{\pi}{L}\right)^2$ and the critical length can be defined as $L_{crit} = \pi\sqrt{\frac{D}{C}}$.

Reflecting boundary conditions

Now, let's analyze what happens for reflecting BC, i.e. $-\rho'(0,t) = 0 = \rho'(L,t)$. The spatial and temporal solutions are:

$$\chi(x) = A\sin\left(\sqrt{\frac{C}{D}-\kappa x}\right) + B\cos\left(\sqrt{\frac{C}{D}-\kappa x}\right)$$

$$T(t) = \exp(\kappa t)$$

and the spatial derivative for the separation of variables solution is:

$$\chi'(x) = A\sqrt{\frac{C}{D}-\kappa}\cos\left(\sqrt{\frac{C}{D}-\kappa x}\right) - B\sqrt{\frac{C}{D}-\kappa}\sin\left(\sqrt{\frac{C}{D}-\kappa x}\right)$$

Applying the BCs, we have:

$$\chi'(0) = 0 \Rightarrow \boxed{A = 0}$$

$$\Rightarrow \chi'(x) = -B\sqrt{\frac{C}{D}-\kappa}\sin\left(\sqrt{\frac{C}{D}-\kappa x}\right)$$

and

$$\chi'(L) = 0 \Rightarrow 0 = -B\sqrt{\frac{C}{D}-\kappa}\sin\left(\sqrt{\frac{C}{D}-\kappa L}\right)$$

Which yields the same constraint as in the absorbing BCs case:

$$\kappa_n = \frac{C}{D} - \left(\frac{n\pi}{L}\right)^2$$

but also $\kappa = \frac{C}{D}$ is a solution that ensures the boundary condition is satisfied. This means that $\chi(x)$ becomes a constant, and it is the first term of the series solution:

$$\chi(x) = B_0 + \sum_{n=1}^{\infty} B_n \cos(\frac{n\pi}{L}x)$$

which satisfies the reflecting boundary conditions. Since $\kappa = \frac{C}{D} > 0$, the temporal solution will always diverge for $t \to \infty$.

(b) **Use a Crank-Nickelson algorithm to numerically compute the neutron density as a function of time up to $T = 10s$, with $C = 1s^{-1}$, $L = 1m$, and $D \in \{1, 10, 100\}cm^2/s$. Comment on similarities or differences you see for the different values of D. Hint: The $C - N$ algorithm is implemented in Garcia's *schro.x*, which you are free to modify.**

We want to implement the C-N algorithm for equation 1. We can rewrite it as:

$$\frac{\rho_j^{n+1}-\rho_j^n}{\Delta t} = \alpha\left(D\frac{\rho_{j+1}^{n+1}-2\rho_j^{n+1}+\rho_{j-1}^{n+1}}{\Delta x^2}+C\rho_j^{n+1}\right)+(1-\alpha)\left(D\frac{\rho_{j+1}^{n}-2\rho_j^{n}+\rho_{j-1}^{n}}{\Delta x^2}+C\rho_j^{n}\right)$$

This can be conveniently rewritten by defining the following matrix and vectors:

$$P_{jk} = D\frac{\delta_{j+1,k}-2\delta_{j,k}+\delta_{j-1,k}}{\Delta x^2}+C\delta_{j,k}$$

$$(\vec{\rho})_j = \rho_j$$

Hence:

$$(\vec{\rho})_j^{n+1} = (\vec{\rho})_j^n + \Delta t\sum_{k=1}^{N} P_{jk}(\alpha\rho_k^{n+1}+(1-\alpha)\rho_k^n)$$

which can be written more compactly as:

$$(\mathbf{I}-\alpha\Delta t P)\vec{\rho}^{n+1} = (\mathbf{I}+(1-\alpha)\Delta t\mathbf{P})\vec{\rho}^n$$

If we let $\alpha = \frac{1}{2}$:

$$(\mathbf{I}-\frac{\Delta t}{2}\mathbf{P})\vec{\rho}^{n+1} = (\mathbf{I}+\frac{\Delta t}{2}\mathbf{P})\vec{\rho}^n$$

Now, we can finally implement the Crank-Nicolson by taking the inverse of the matrix in the LHS:

$$\vec{\rho}^{n+1} = \left(\mathbf{I}-\frac{\Delta t}{2}\mathbf{P}\right)^{-1}\left(\mathbf{I}+\frac{\Delta t}{2}\mathbf{P}\right)\vec{\rho}^n$$

I modified the program schro.m to implement this algorithm in Matlab. I will describe in detail the structure of the program for Neumann boundary conditions(BCs) below but just briefly comment the difference with the others.

Neumann boundary conditions

Neumann BCs can be implemented as follows:

$$\frac{\partial\rho}{\partial x}|_{bdry} = \frac{\rho_{j+1}^n-\rho_{j-1}^n}{2\Delta x} \approx 0$$

$$\Rightarrow \rho_{j+1}^n = \rho_{j-1}^n$$

for j=1 and j=N-1.

For the program, we first define the parameters for our system:

```
%% * Initialize parameters (grid spacing, time step, etc.)
Nprime = 100;    %input('Enter number of grid points: ');
N=Nprime+2;      %adding the ghost cells to apply Neumann BCs.
L = 100;         % System extends from -L/2 to L/2
delx = L/(N-1-2); % Grid cell size
x = delx*(0:N-1) - delx*(N-1)/2;  % Coordinates  of grid points
D = input('Enter D: ');  % diffusion coefficient. Larger means faster diffusion
C=1;             % rate of neutron creation processes
tau = 0.01;      %input('Enter time step: ');
```

where we add two cells to the grid to act as ghosts and ensure Neumann BCs are correctly applied. Then, we initialize the diffusion matrix **P** and the C-N matrix:

```matlab
%% * Set up the Diffusion operator matrix
Diff = zeros(N);  % Set all elements to zero
coeff = D/(delx^2);
for i=2:(N-1)
  Diff(i,i-1) = coeff;
  Diff(i,i) = -2*coeff+C;  % Set interior rows
  Diff(i,i+1) = coeff;
end
% First and last rows for Neumann BCs
Diff(1,:) = Diff(3,:);
Diff(N,:) = Diff(N-2,:);

%% * Compute the Crank-Nicolson matrix
dCN = ( inv(eye(N) - .5*tau*Diff) * (eye(N) + .5*tau*Diff) );
```

Then, we initialize the density function at $t = 0$, that is, the initial shape of the neutron density.

```matlab
%% * Initialize the density function at t=0
%I chose a Gaussian wave packet as the starting function, but a ↵
%function will serve as well
x0 = 0;                    % Location of the center of the wavepacket
sigma0 = L/20;             % Standard deviation of the wavefunction
Norm_rho = 1/(sqrt(sigma0*sqrt(pi)));  % Normalization
Rho = Norm_rho * exp(-(x'-x0).^2/(2*sigma0^2)); %to add velocity .*exp(i_imag*k0*x')
```

Finally, we implement the C-N iterative algorithm until we reach a fixed number of iterations:

```matlab
%% * Loop over desired number of steps
for iter=1:max_iter
  %* Compute new density function using the Crank-Nicolson scheme
  Rho = dCN*Rho;
end
```

The rest of the code is just generating the different plots. I will avoid that discussion, since we have used plotting extensively in the previous handouts. The resulting plots for $D = 1cm^2/s$ are shown in figure 1

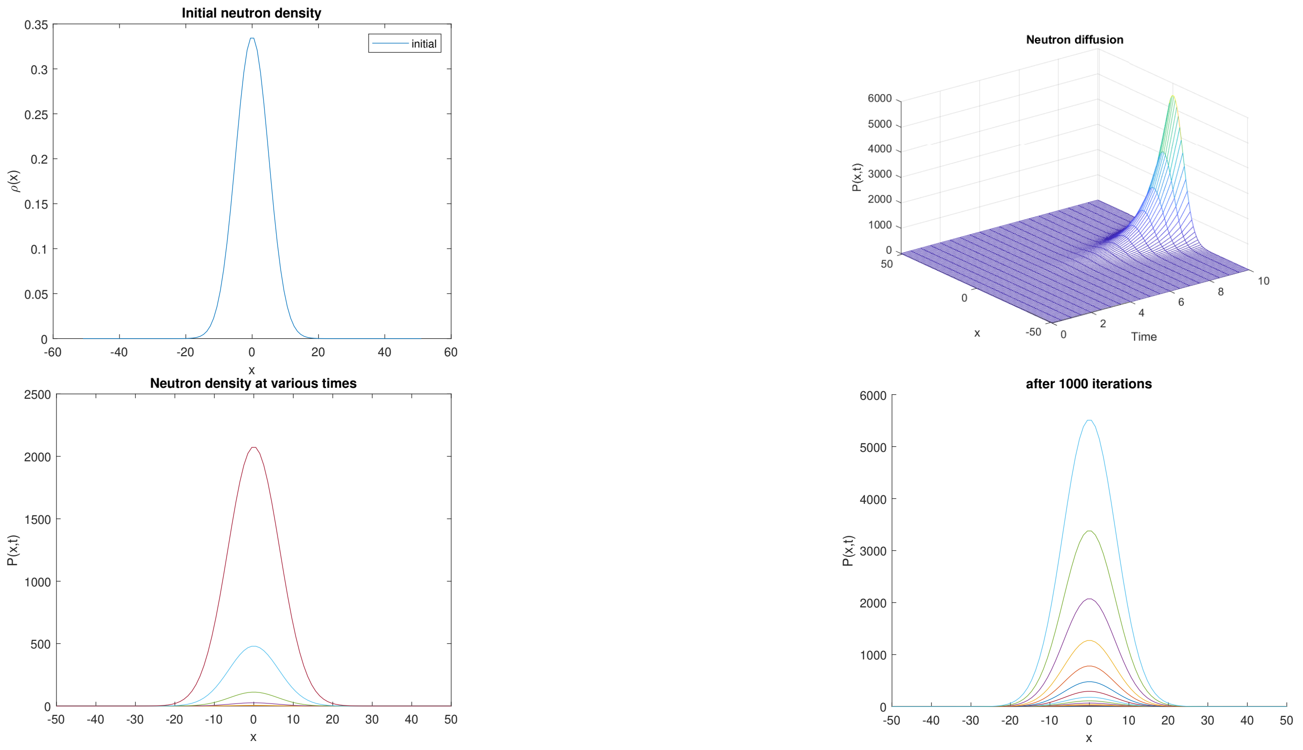


Figure 1: Solutions for $D = 1cm^2/s$ with reflecting BCs.

It can be seen that as the diffusion coefficient increases, the neutron density grows faster and faster in the same time interval. Also, the behavior at the boundary is the expected with the derivatives at those points clearly being zero. It must be noticed that the problem requested $L = 1m$ and $D = 1e{-}4, 1e{-}3, 1e{-}2cm^2/s$ for the parameters. In my code I used $L = 100$ and $D = 1, 10, 100$. However, since the coefficient in the differential equation is given by $coeff = \frac{D}{(\Delta x)^2} \propto \frac{D}{L^2} \Rightarrow \frac{1}{100^2} = 1e - 4$, $\frac{10}{100^2} = 1e - 3$, and $\frac{100}{100^2} = 1e - 2$ the results are equivalent as the adimensional constant is the same. Still, the difference will be shown is the plots as a different scaling on the x axis, equivalent to showing the "units" in cm instead of m. Figures 2-3 show the solutions for $D = 10cm^2/s$ and $D = 100cm^2/s$.

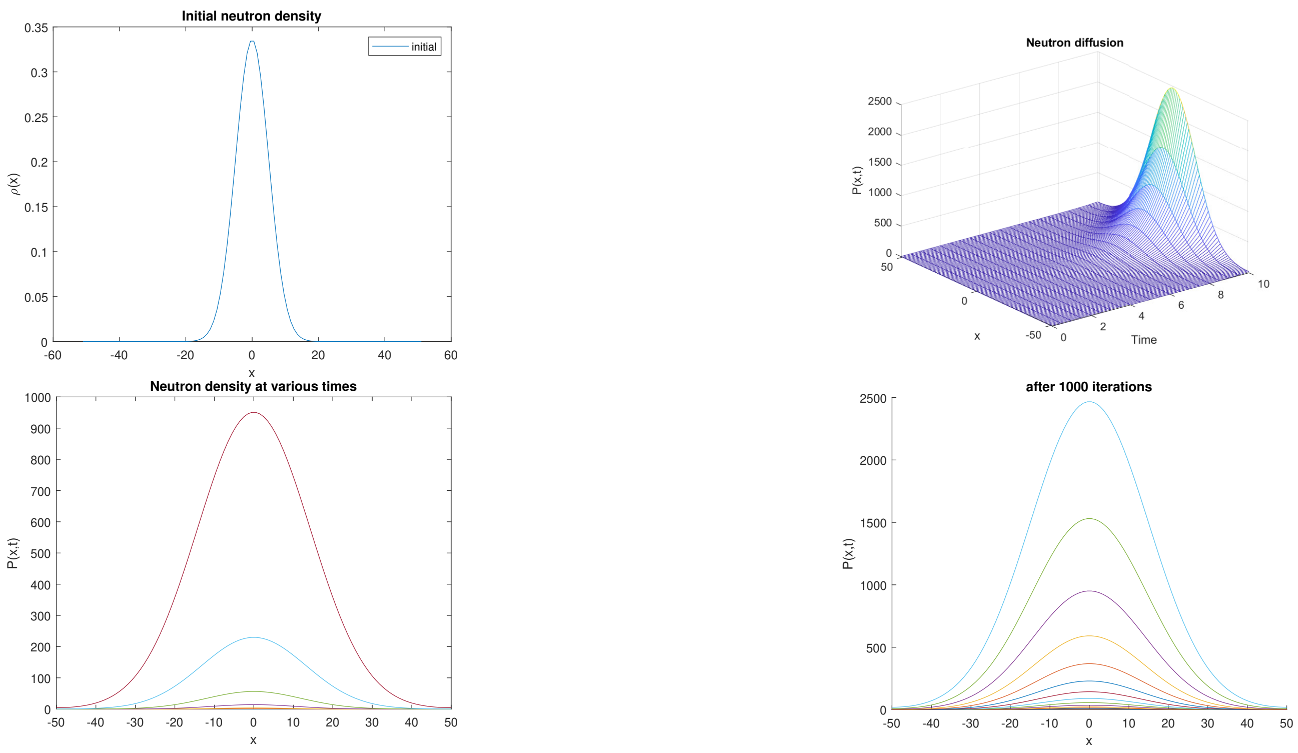


Figure 2: Solutions for $D = 10cm^2/s$ with reflecting BCs.

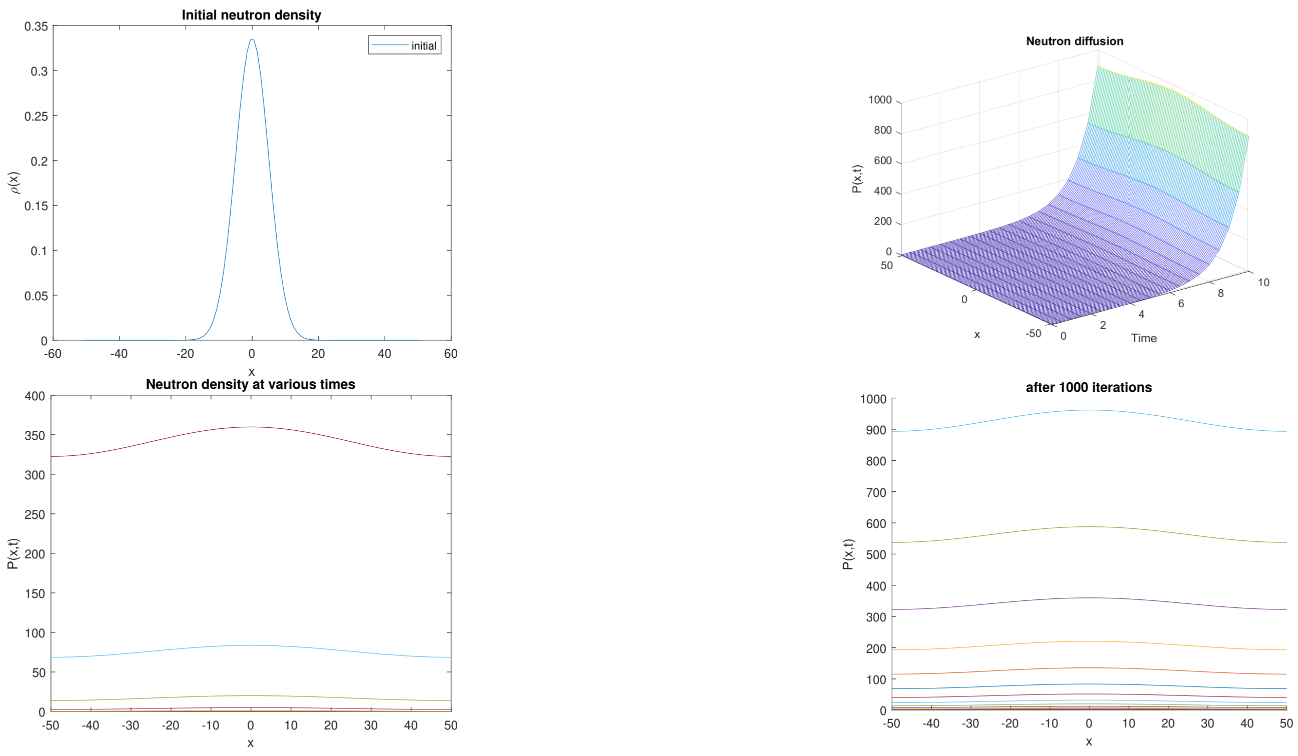


Figure 3: Solutions for $D = 100cm^2/s$ with reflecting BCs.

Periodic boundary conditions

The behavior is pretty much similar to the one for Neumann BCs as it is shown in figures 4-6.

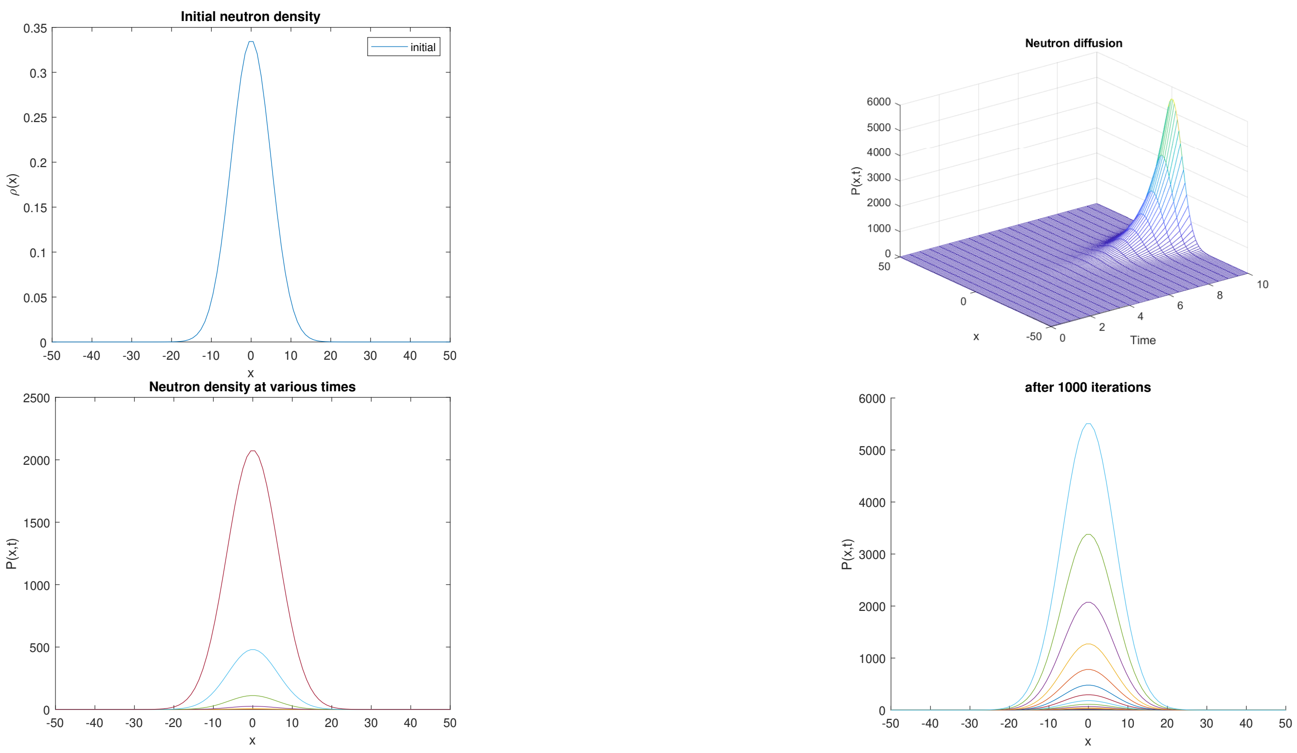


Figure 4: Solutions for $D = 1cm^2/s$ with periodic BCs.

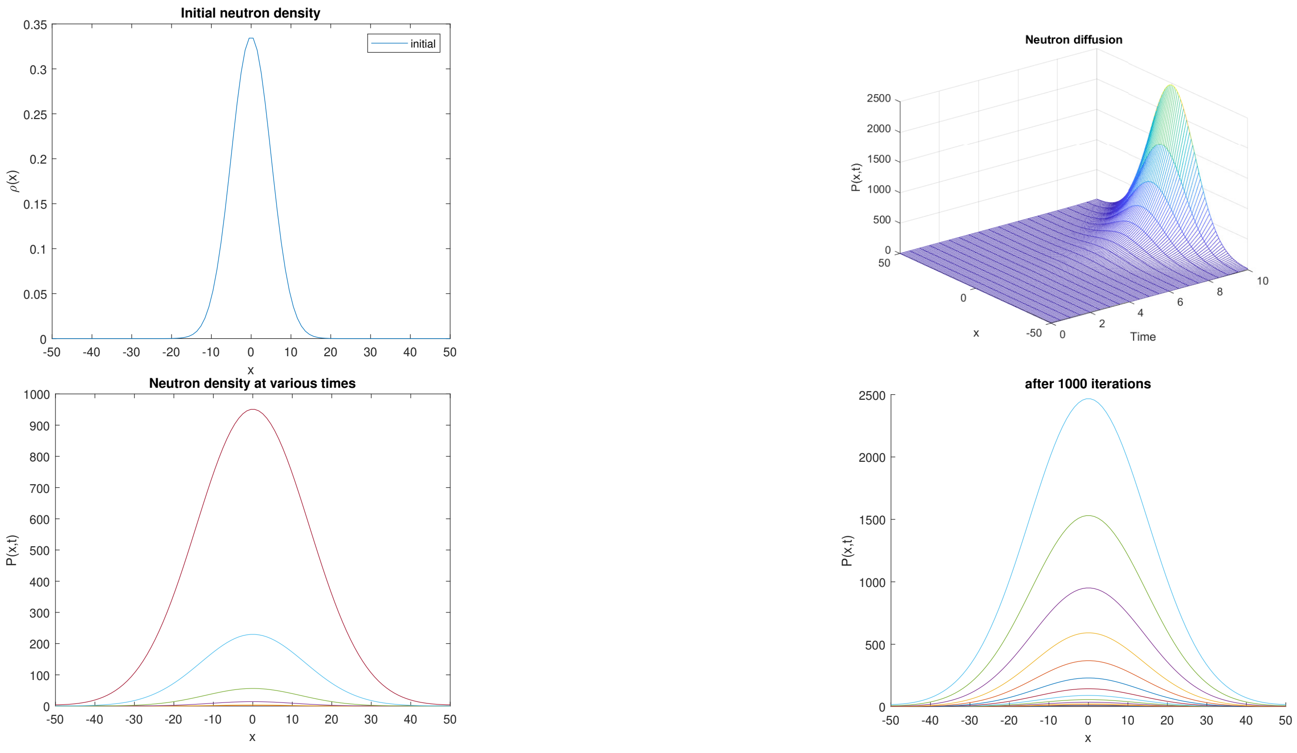


Figure 5: Solutions for $D = 10cm^2/s$ with periodic BCs.

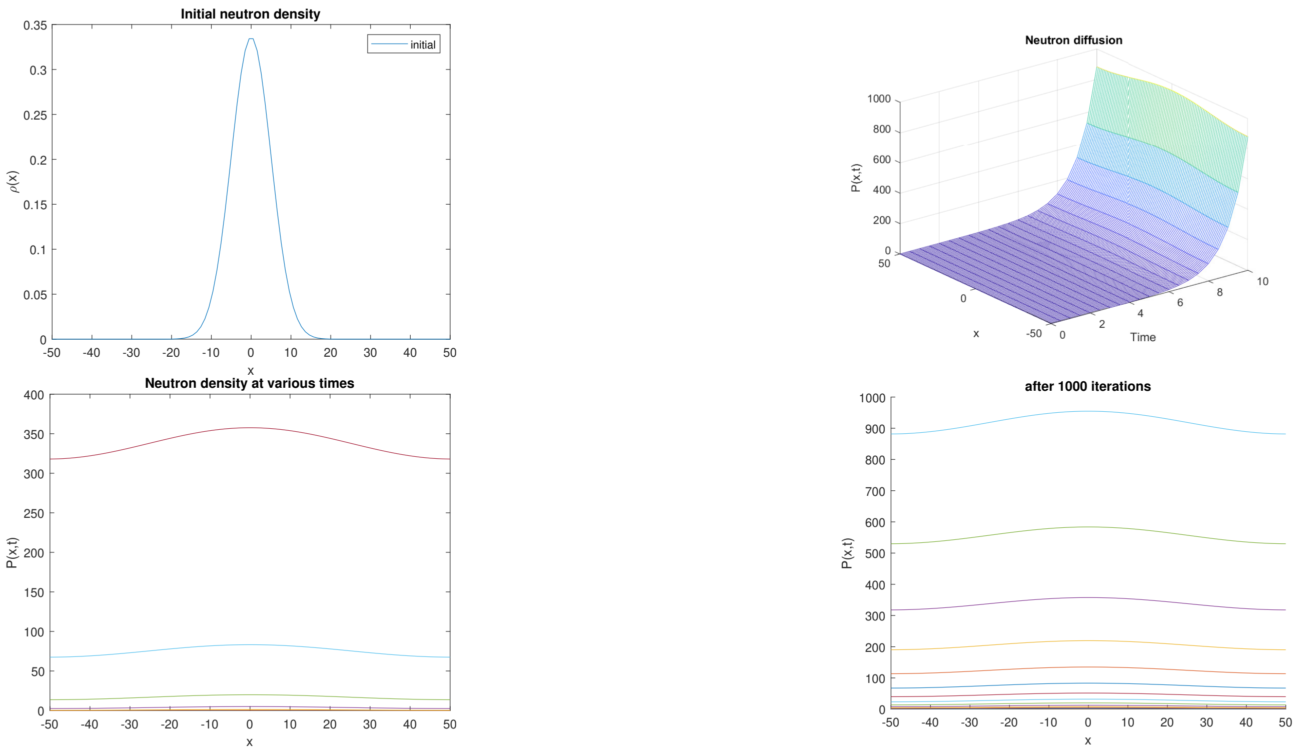


Figure 6: Solutions for $D = 100cm^2/s$ with periodic BCs.

Dirichlet boundary conditions

In this case, the behavior totally differs from the previous cases, since we now enforce the value at the boundaries to be zero. Figures 7-9 show the results.

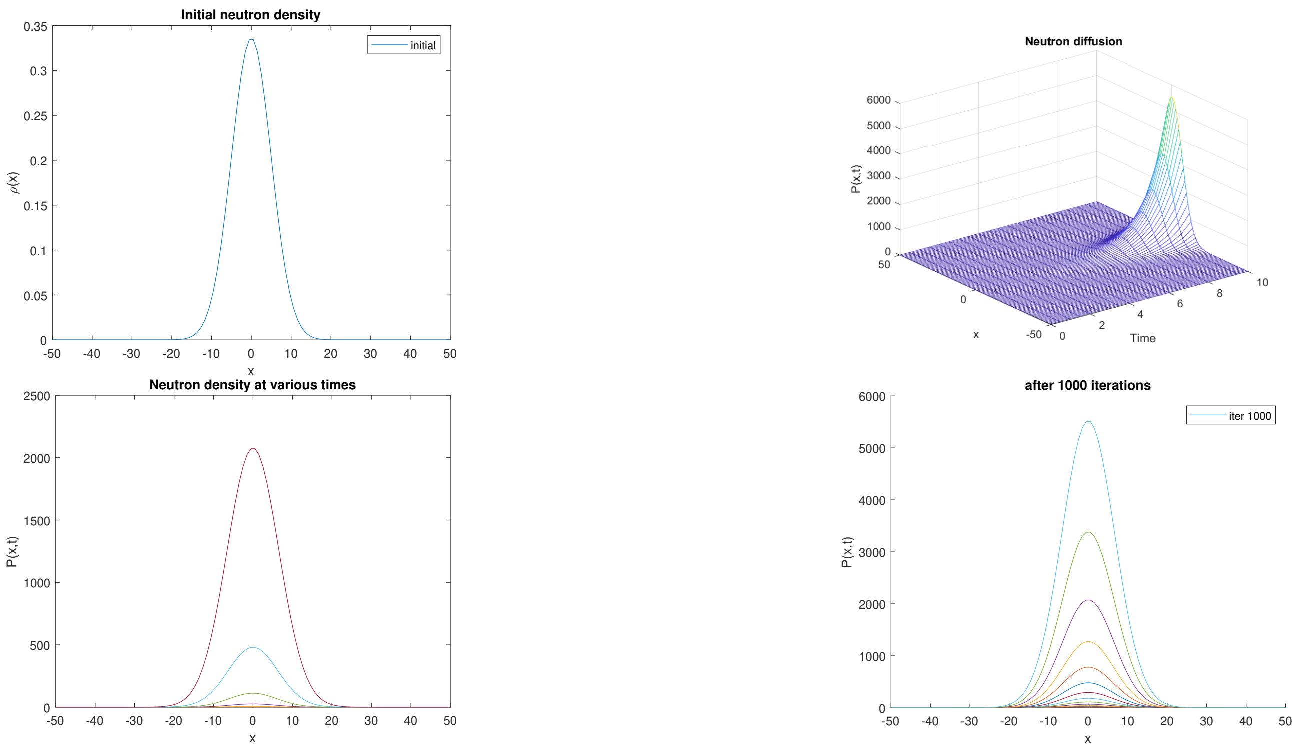


Figure 7: Solutions for $D = 1cm^2/s$ with Dirichlet BCs.

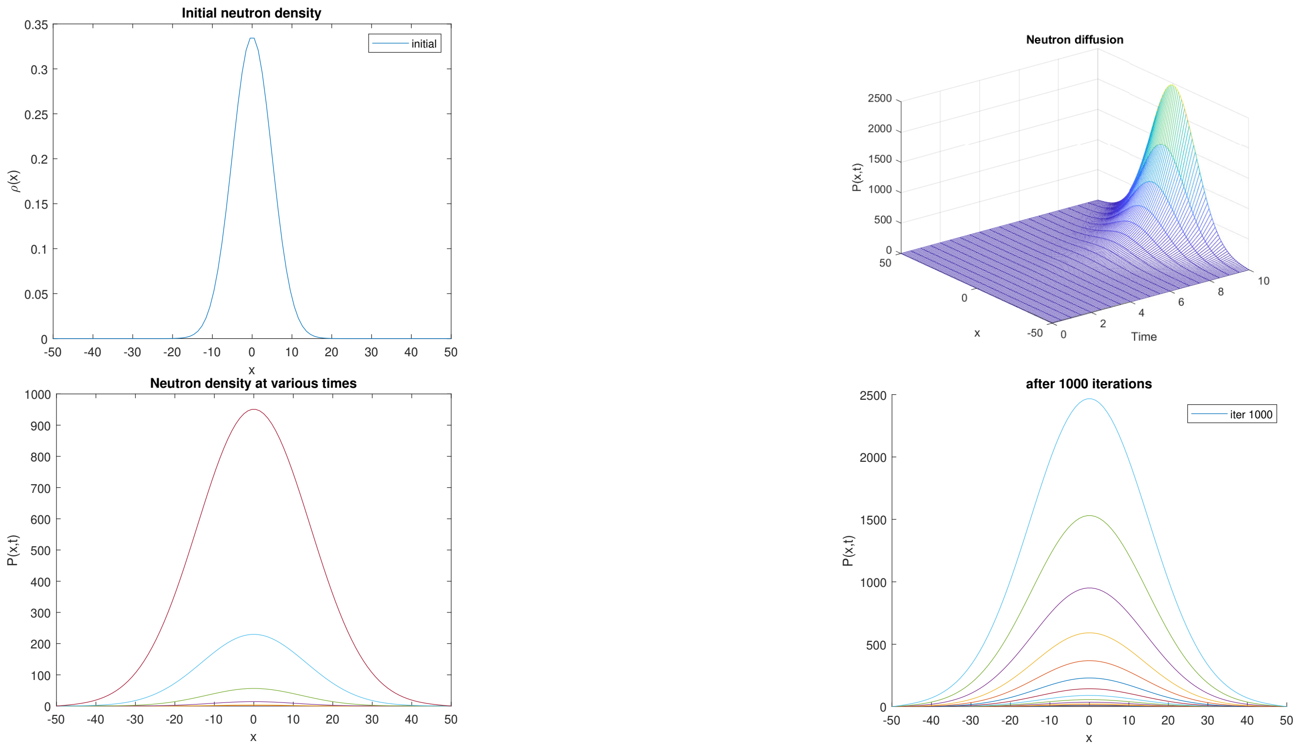


Figure 8: Solutions for $D = 10cm^2/s$ with Dirichlet BCs.

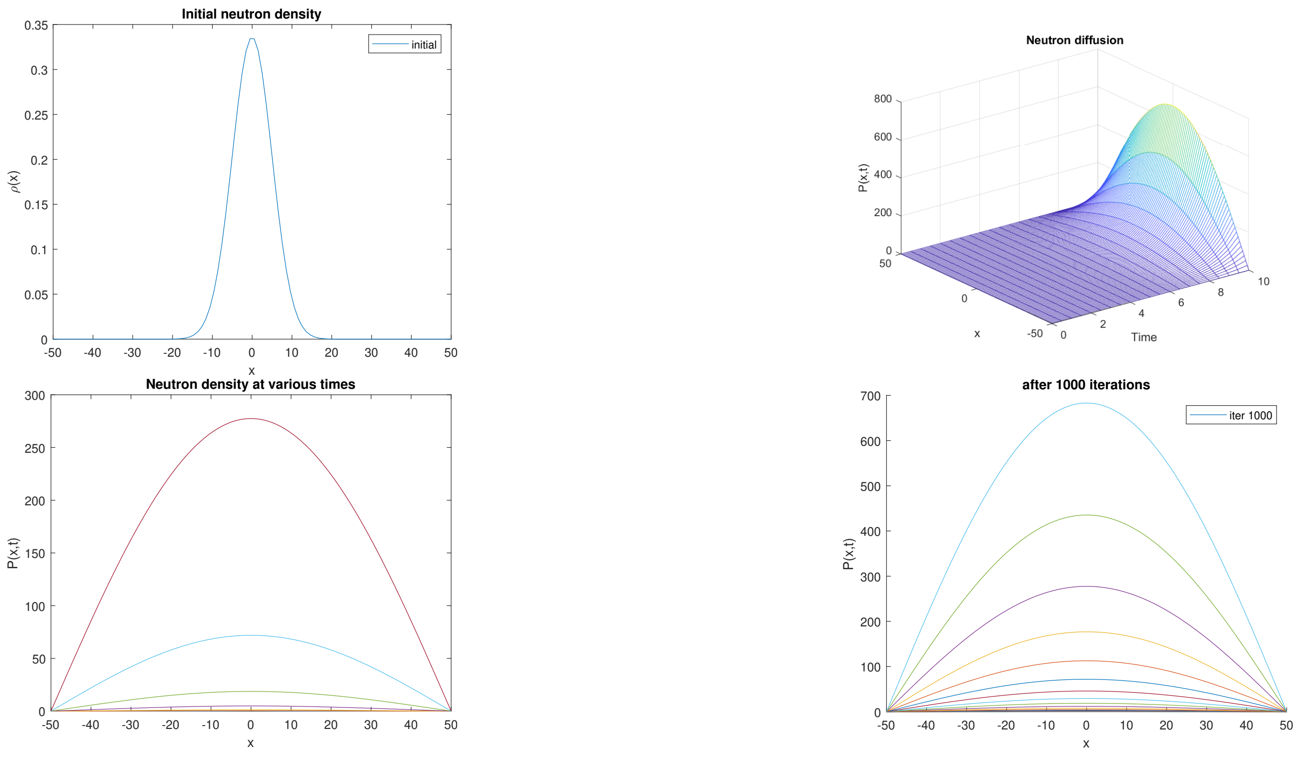
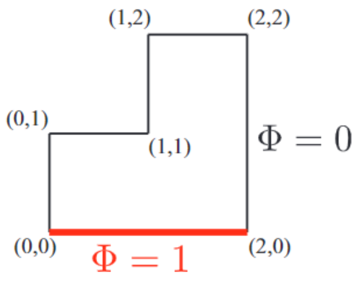


Figure 9: Solutions for $D = 100cm^2/s$ with Dirichlet BCs.

2 Problem 2

In the domain sketched below, the black thin walls all have $\phi = 0$ while the thick red wall has $\phi = 1$.



(a) **Numerically solve Laplace's equation $\nabla^2\phi = 0$ with these boundary conditions. You may use any method you would like, but please describe the method you chose, including initial conditions, timestep, grid points, etc for finite difference methods, or number of modes for spectral methods. Show a contour plot for your solution.**

Boundary conditions given on irregular geometries pose a great challenge to solve a problem numerically. In this case, since we only have a combination of square geometries, the simplest approach, by far, is to consider a full square of side 2. Thus, we are adding some ghost cells to our problem, which can be dealt by fixing the values of the potential inside those ghost cells to 0. I chose to use overrelaxation methods, in which we consider a time dependent problem, but wait long enough to let the system achieve the steady state. In this way, we remove the unwanted time dependency. The program structure was modified from the relax.m provided by Gracia. First, we set up the parameters for our system:

```
%% * Initialize parameters (system size, grid spacing, etc.)
method = menu('Numerical Method','Jacobi','Gauss-Seidel','SOR');
N = input('Enter number of grid points on a side: ');
L = 1;            % System size (length)
h = L/(N-1);      % Grid spacing
x = 2*(0:N-1)*h;  % x coordinate
y = 2*(0:N-1)*h;  % y coordinate
```

For this algorithms, we need to provide an initial guess for the values of the potential. One must be a little bit careful since a particular bad guess will increase the computational time incredibly. We also need to set up the BCs:

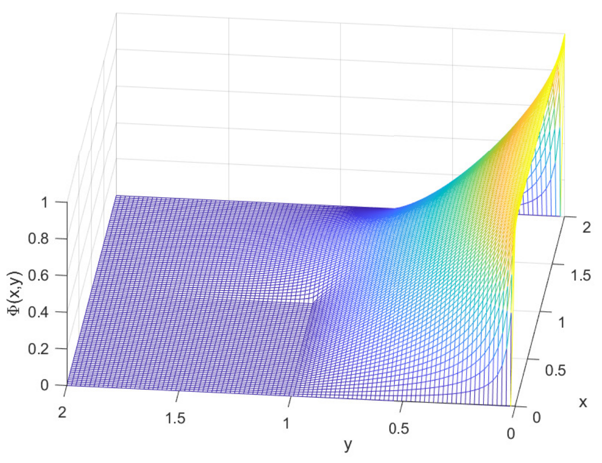
```
%% * Set initial guess as first term in separation of variables soln.
phi0 = 1;      % Potential at y=L
phi =1% phi0 * 4/(pi*sinh(pi)) * sin(pi*x'/L)*sinh(pi*y/L);

%% * Set boundary conditions
phi(round(N/2)+1:N,1:round(N/2)) = 0;   phi(:,1) = 0;   phi(:,N) = 0;
phi(1,:) = phi0*ones(N,1);
fprintf('Potential at y=0 equals %g \n',phi0);
fprintf('Potential is zero on all other boundaries\n');
```

Then, we implemented the iterative part of the algorithm until we reach the steady state. That is, we iteratively compute phi until the change between consecutive calculations is below a fixed tolerance:

```
%% * Loop until desired fractional change per iteration is obtained
```

```
newphi = phi;            % Copy of the solution (used only by Jacobi)
iterMax = N^2;           % Set max to avoid excessively long runs
changeDesired = 1e-4;    % Stop when the change is given fraction
fprintf('Desired fractional change = %g\n',changeDesired);
tStart = cputime;        % Start the stopwatch
for iter=1:iterMax
  changeSum = 0;

  if( method == 1 )       %% Jacobi method %%
    for i=2:(N-1)         % Loop over interior points only
     for j=2:(N-1)
        if i>round(N/2) && j<round(N/2)
            newphi(i,j)=0;
        else
            newphi(i,j) = .25*(phi(i+1,j)+phi(i-1,j)+ ...
                                  phi(i,j-1)+phi(i,j+1));
        changeSum = changeSum + abs(1-phi(i,j)/newphi(i,j));
        end
     end
    end
    phi = newphi;

  elseif( method == 2 )  %% G-S method %%
    for i=2:(N-1)         % Loop over interior points only
     for j=2:(N-1)
        if i>round(N/2) && j<round(N/2)
            newphi=0;
        else
            newphi = .25*(phi(i+1,j)+phi(i-1,j)+ ...
                                   phi(i,j-1)+phi(i,j+1));
            changeSum = changeSum + abs(1-phi(i,j)/newphi);
        end
    phi(i,j) = newphi;
     end
    end

  else                    %% SOR method %%
    for i=2:(N-1)         % Loop over interior points only
     for j=2:(N-1)
       if i>round(N/2) && j<round(N/2)
            newphi=0;
       else
            newphi = 0.25*omega*(phi(i+1,j)+phi(i-1,j)+ ...
               phi(i,j-1)+phi(i,j+1))  +  (1-omega)*phi(i,j);
            changeSum = changeSum + abs(1-phi(i,j)/newphi);
       end
       phi(i,j) = newphi;
     end
    end
  end

  %* Check if fractional change is small enough to halt the iteration
  change(iter) = changeSum/(N-2)^2;
  if( rem(iter,10) < 1 )
    fprintf('After %g iterations, fractional change = %g\n',...
                             iter,change(iter));
  end
  if( change(iter) < changeDesired )
    fprintf('Desired accuracy achieved after %g iterations\n',iter);
    fprintf('Breaking out of main loop\n');
    break;
  end
end
tStop = cputime;     % Stop the stopwatch
```

The output plots are shown in figure 10.



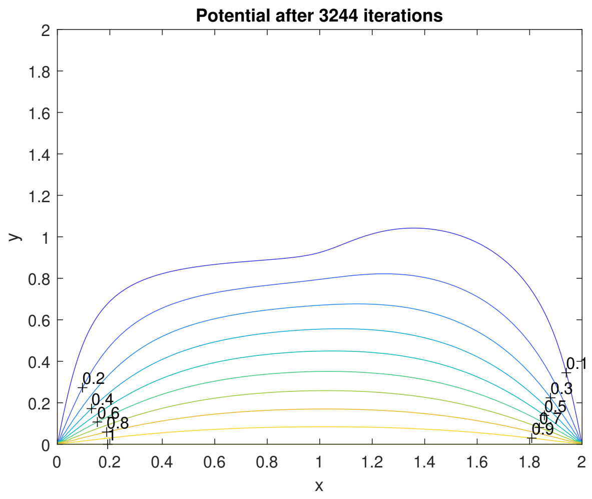


Figure 10: Solutions for Laplace equation with the given BCs geometry.

(b) **Use the same method to solve Poisson's equation, $\nabla^2\phi = -\rho(r)/\epsilon_0$, with a point charge located at (0,1/2). Plot a contour plot of the solution.**

The only difference in the code is in the initial conditions and in an added term into the loop that accounts for the presence of a charge density, $1/\epsilon_0 * h^2 * \rho(i,j)$:

```
      %% * Set boundary conditions
phi(round(N/2)+1:N,1:round(N/2)) = 0;   phi(:,1) = 0;   phi(:,N) = 0;
phi(1,:) = phi0*ones(N,1);
\textbf{Rho = zeros(N,N);
[ ¬, iy ] = min( abs( y-0.5 ) ); %find the index of cell closest to y=0.5
[ ¬, ix ] = min( abs( x ) );      %find the index of cell closest to x=0
Rho(iy,ix)=1/h;
phi=phi+Rho;
}
```

```
      %modification for the Jacobi method, all others are analogous
      for i=2:(N-1)         % Loop over interior points only
      for j=2:(N-1)
          if i>round(N/2) && j<round(N/2)
              newphi(i,j)=0;
          else
              newphi(i,j) = .25*(phi(i+1,j)+phi(i-1,j)+ ...
                                   phi(i,j-1)+phi(i,j+1))+ 1/eps0*h^2*Rho(i,j);
          changeSum = changeSum + abs(1-phi(i,j)/newphi(i,j));
          end
      end
     end
     phi = newphi;
```

The only difference with the previous result is the presence of the infinite potential due to the point charge. As depicted in figure 11, the value of the potential due to the point charge completely clouds the potential due to the line of charge at $y = 0$. Recall, that the potential for a point charge is infinite.

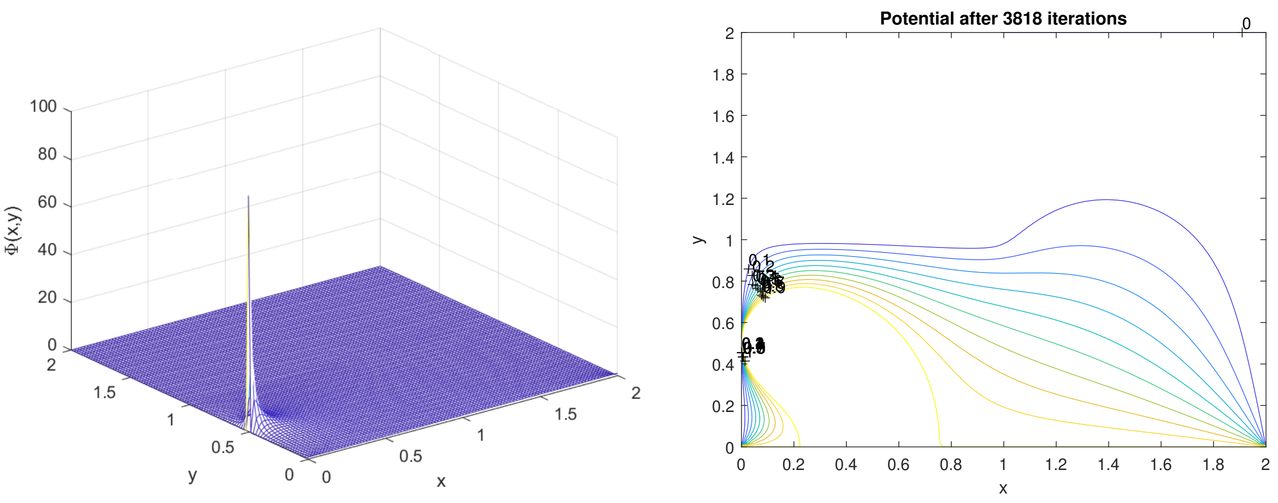


Figure 11: Solutions for Poisson equation with the given BCs geometry and charge density due to a point charge.