Boundary Problem of Electrostatics

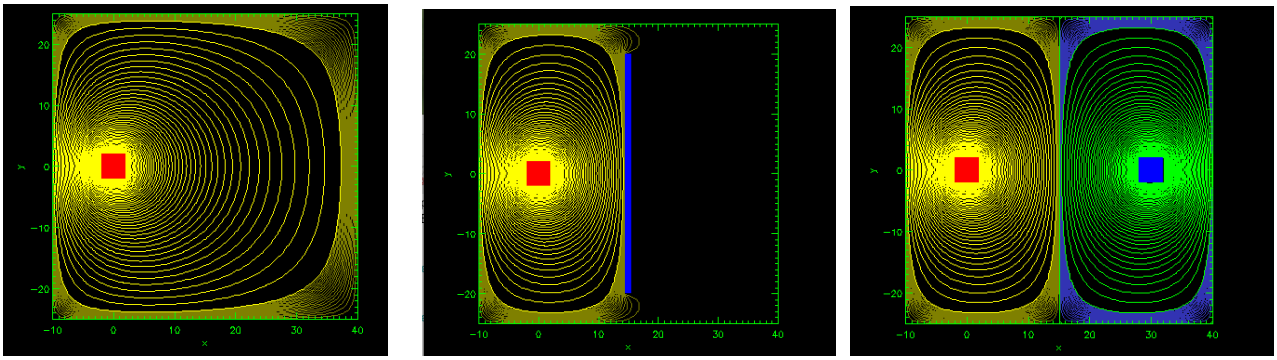
May 2019

1 Electric potential of three different cases

Using the relaxation method explained in the problem sheet, we can get the potential distribution of three different cases, where we used random initial condition and do iteration over 10000 times with algorithm:

$$\phi_{ij}^{k+1} = (\phi_{i+1j}^{k} + \phi_{i-11}^{k} + \phi_{ij+1}^{k} + \phi_{ij-1}^{k})/4 \tag{1}$$

In addition, over-relaxation parameter $w = 1.95$ is used to speed up the process. Other parameters: $gridsize$: $dx = dy = 0.5$, $boundaries = 0$. The results are showed in the Figure.1 (the code is attached at the end):



(a) Case A. (b) Case B. (c) Case C.

Figure 1: Potential distribution of three different cases

As we can see, case B and case C look almost the same in terms of the left side. This is because in case C the negative anode on the right side is the mirror charge of the left one. But we can still see a slight difference around the both ends of the metal. This is due to finite length of the metal in the middle.

For the fixed point, we can see the potential value changes over steps. In all three cases, the potential converges to a finite value after about 2000 iterations (Figure. 2). We find the values for three cases respectively

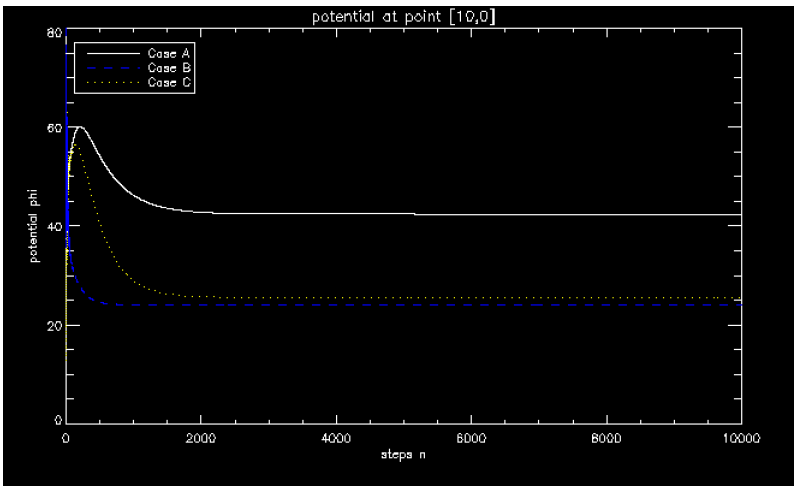


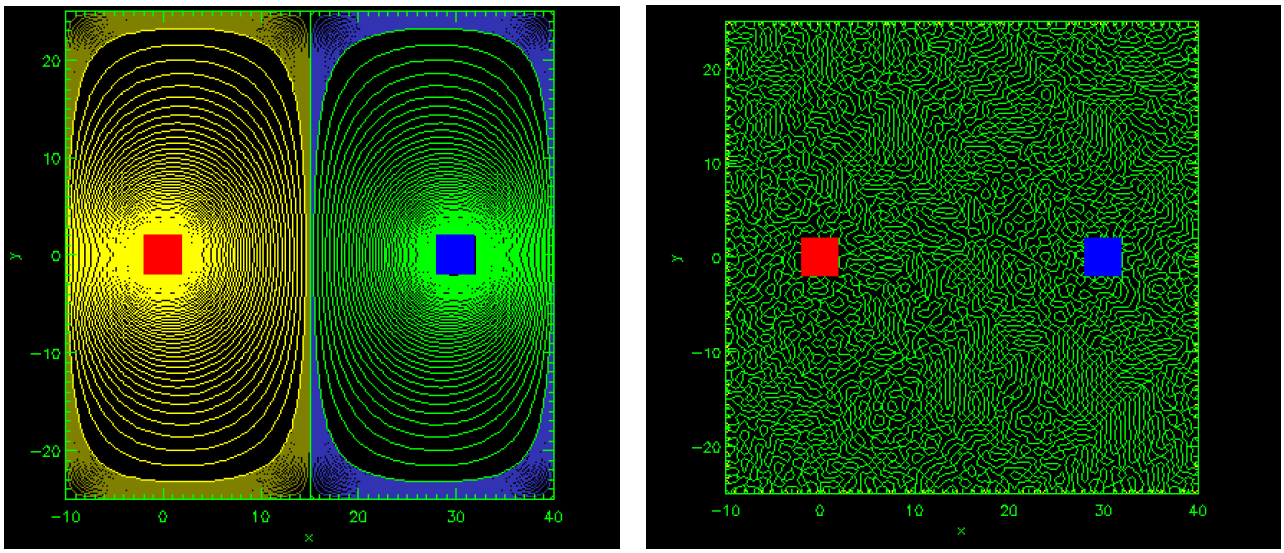
Figure 2: Potential at point [10,0] changes over steps

as follows: 42.430917, 24.130342, 25.566845. The final potential at [10,0] is quite close to each other in case B and C, which is what we expect.

2 Further Discussion

2.1 Discussion on over-relaxation parameter

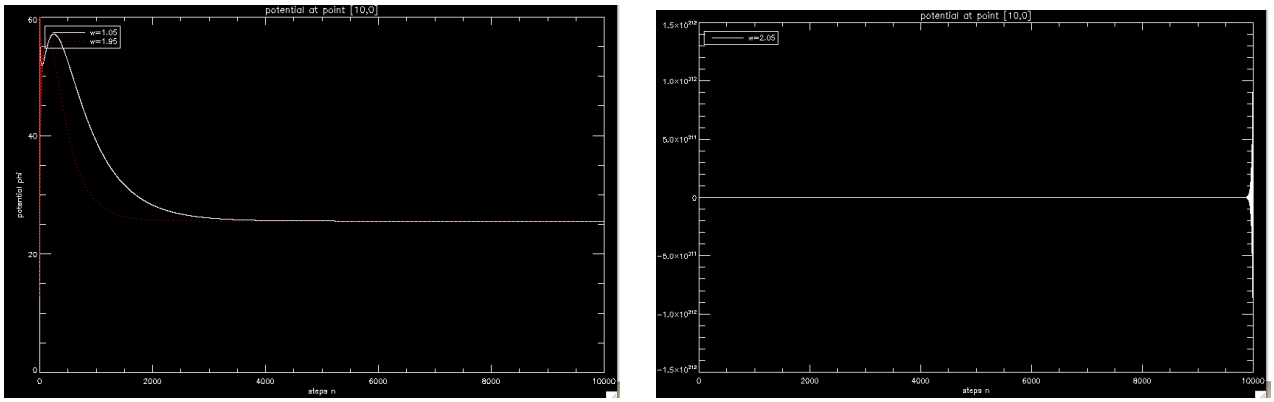
We used over-relaxation parameter to accelerate the iteration process. To get a reliable result, obviously w cannot be infinitely large. In the following, three different values 1.05, 1.95(see Figure 1.c), 2.05 are tried and we can see from the potential distribution plots that when $w = 2.05$ the whole process breaks down(Figure. 3).



(a) w=1.5 (b) w=2.05

Figure 3: Influence of over-relaxation parameter

And when $w > 2$, the potential value at point [10,0] explodes after enough steps(Figure. 4).



(a) w=1.5 (b) w=2.05

Figure 4: Influence of over-relaxation parameter

In fact, for convergence it is originally required that $0 < w < 2$. And for squared grid as we have in this case, the optimum value of w is $w_{opt} \approx \frac{2}{1+\pi/\sqrt{M}} \approx 1.94$ (M is the number of grids in one dimension).

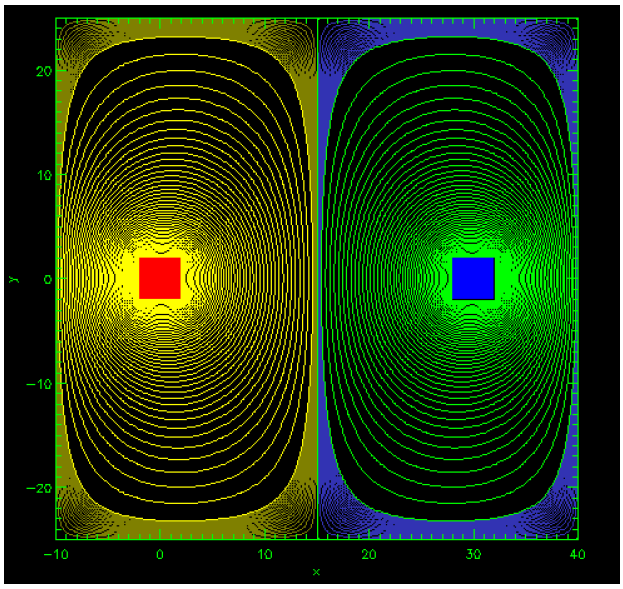
2.2 Discussion on grid size

At first we set grid size as $0.5 * 0.5$, what if we choose smaller sizes? Would the result be more accurate?

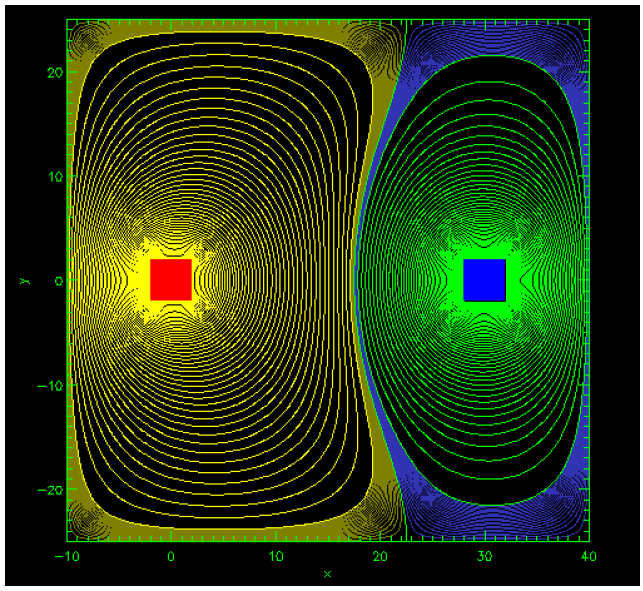
With grid size $0.25 * 0.25$, the result doesn't seem very different. But if we choose a even smaller size, for example, $0.125 * 0.125$, the potential plot seems quite different and unphysical(Figure. 5 b).

The reason may be if we cut the plane in more detail, it would decrease the sensitivity between two grids which are far from each other. In principle, this could be solved by simply do more iterations. For example, if we do 30000 iterations rather than 10000, then the result looks 'normal' again(Figure 6).

Figure 7 shows the potential difference(in natural logarithm) changes over steps of three different grid size cases('$dx = dy = 0.5$','$dx = dy = 0.25$','$dx = dy = 0.125$'). We can see from that all three lines come down



(a) dx=dx=0.25



(b) dx=dy=0.125

Figure 5: Influence of grid size

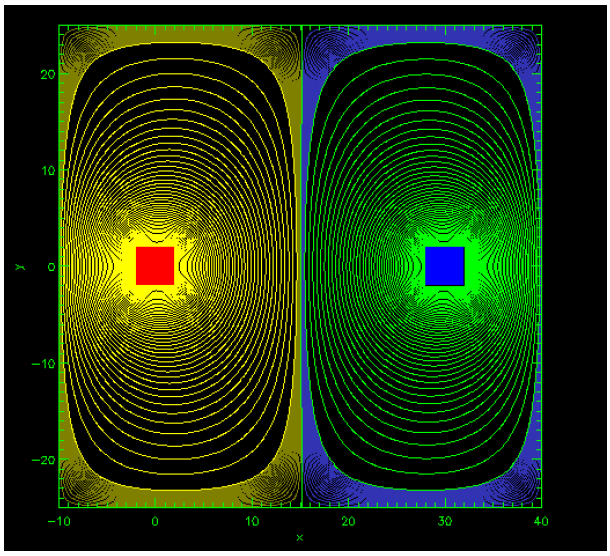


Figure 6: Influence of grid size, dx=dy=0.125, 30000 steps

linearly which means $\Delta\phi$ converges when the steps go to infinity, since it implies $\Delta\phi_{i+1} = \alpha\Delta\phi_{i+1}$ with $\alpha \ll 1$. But in '$dx = dy = 0.5$' case, it drops fastest of all. That's why it takes less iterations to reach the stable solution compare to the case where '$dx = dy = 0.125$'.

2.3 Discussion on periodic boundaries

At first, we set the potential at the boundaries to zero. How the result would look like if we choose periodic boundary conditions. Since in the equation (1), we actually already used the periodic information when we calculated the points on the boundary. We just need to undo the boundary set over the whole iteration, then we would automatically have potential distribution for periodic boundaries(Figure. 7).

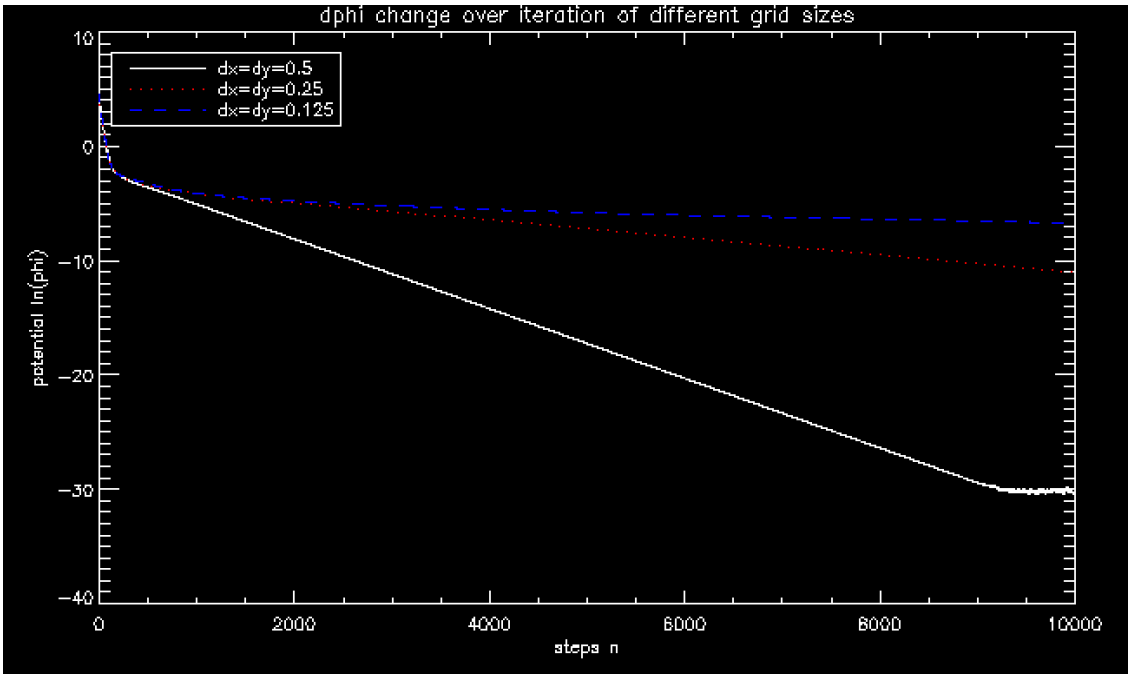


Figure 7: Maximal potential difference over steps in different unit scales

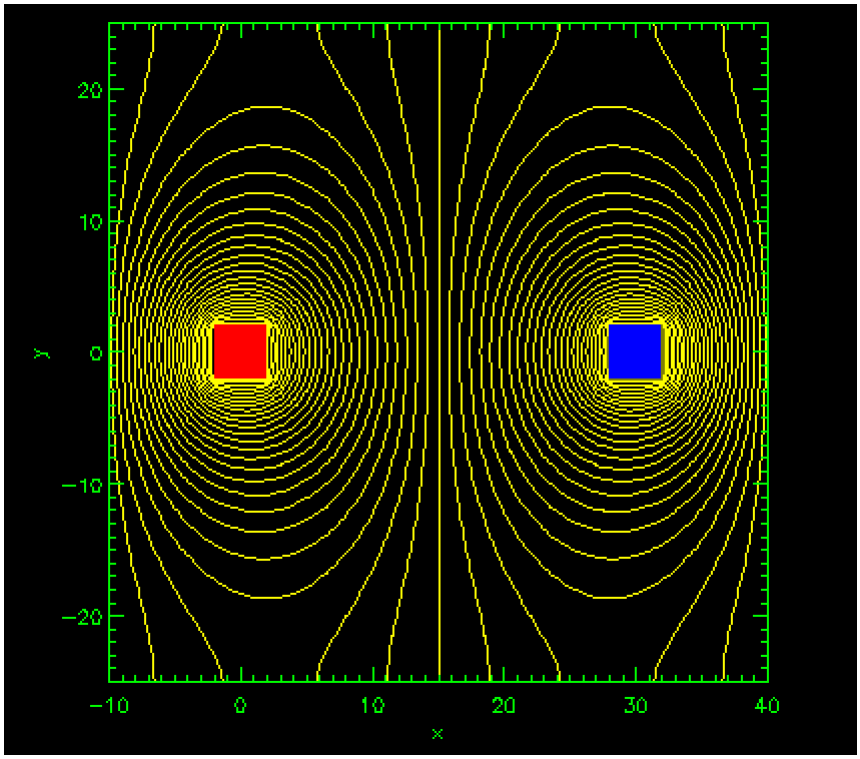


Figure 8: Case C with periodic boundary

```
;to calculate the potential with 'for' loop
pro laplace_iter, niter, w, phianode, anode, phiblende, blende, phirand, rand, phi, dphi,
  ;niter: times of iterations
  wm = 1.0-w
  dphi = dblarr(niter,/nozero) ;potential difference over two iterations
  phix = dblarr(niter,/nozero) ;potential at fixed point [10,0]
  for i=0,niter-1 do begin
     d = 0.25*(shift(phi,1,0) + shift(phi,0,1) + shift(phi,-1,0) + shift(phi,0,-1))
     laplace_setbc, phianode, anode, phiblende, blende, phirand, rand, d
     d = 0.25*(shift(d,1,0) + shift(d,0,1) + shift(d,-1,0) + shift(d,0,-1))
     laplace_setbc, phianode, anode, phiblende, blende, phirand, rand, d
     dphi[i]=max(abs(phi-d))
     if (i mod 100 eq 0) then print, niter-i, dphi[i]
     phi = w*d + wm*phi
     phix[i] = phi[40,50]
  endfor
end

;to set values for fixed potentials like potential of anode = 100
pro laplace_setbc, val1, mask1, val2, mask2, val3, mask3, phi
  phi[mask1] = val1
  phi[mask2] = val2
  phi[mask3] = val3
end

;to define different areas in the plane
pro laplace_setmask, sx, x, sy, y, casei, mask1, mask2, mask3
  nx = n_elements(x)
  ny = n_elements(y)
  xx = rebin(x, nx, ny)
  yy = transpose(rebin(y, ny, nx))
  mask1 = where(xx ge -2.0 and xx le 2.0 and  yy ge -2.0 and yy le 2.0)
  mask3 = where(xx eq max(x) or xx eq min(x) or yy eq max(y) or yy eq min(y))
  blend = where(xx ge  14.5 and xx le  15.5  and yy ge  -20.0 and yy le 20.0)
  node2 = where(xx ge  28.0 and xx le  32.0  and yy ge  -2.0 and yy le 2.0 )
  case casei of
     'A': mask2=mask3
     'B': mask2=blend
     'C': mask2=node2
  endcase
end

;to set colors showed in the plots
pro laplace_set_colors, red, yellow, blue, grey, white, green, orange
  red    = 255L
  yellow = 65535L
  blue   = 16711680L
  grey   = 11744050L
  white  = 16777215L
  green  = 65280L
  orange = 32895L
end

;main procedure
pro laplace,niter,w,dx,dy,casei,v1,v2,v3,dphi,phix
  laplace_set_colors, red, yellow, blue, grey, white, green,orange
  sx  = [-10.0, 40.0]
  sy  = [-25.0, 25.0]
  nx  = nint((sx[1]-sx[0])/dx)+1
  ny  = nint((sy[1]-sy[0])/dy)+1
```

```
  x     = sx[0]+findgen(nx)*dx
  y     = sy[0]+findgen(ny)*dy

  phi = dblarr(nx,ny) + randomu(iseed,nx,nx)*100.0
  phiB1  = v1
  phiB2  = v2
  phiB3  = v3

  laplace_setmask, sx, x, sy, y,casei,B1,B2,B3
  laplace_setbc, phiB1,B1,phiB2,B2,phiB3,B3, phi
  laplace_iter,niter, w,phiB1,B1,phiB2,B2,phiB3,B3,phi,dphi,phix

  ;starts to plot
  levels=findgen(21)/20.0*2.0
  contour, phi, x, y, xstyle=5, ystyle=5, levels=levels, /iso, color=orange

  levels=findgen(21)/20.0*2.0-2
  contour, phi, x, y, xstyle=5, ystyle=5, levels=levels, /iso, $
           color=grey, /noerase

  levels=findgen(51)*2.0
  contour, phi, x, y, /xsty, /ysty, levels=levels, /noerase, /iso,color=yellow, $
           xtitle='x', ytitle='y'
  levels=findgen(51)*2.0-100
  contour, phi, x, y, /xsty, /ysty, levels=levels, /noerase, /iso,color=green, $
           xtitle='x', ytitle='y'

  polyfill, [-2, 2, 2, -2, -2], [-2, -2, 2, 2, -2], color=red
  case casei of
     'A' :break
     'B' :polyfill,[14.5,15.5,15.5,14.5,14.5],[-20,-20,20,20,20],color=blue
     'C' :polyfill,[28,32,32,28,28],[-2,-2,2,2,-2],color=blue
  endcase

end
```