Solution to Problem 1

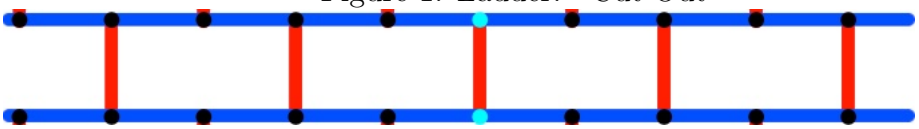
Siddharth Tiwary

January 3, 2018

1 Analytically, The Upper and Lower Bounds

If we cut off some wires from the circuit, the equivalent resistance can only increase, as can be felt intuitively, and as presented in the Electric Circuits booklet. On the other hand, if we short circuit some nodes, it can only decrease. We use these to find our upper and lower bounds.
First, consider the one dimensional infinite ladder containing AB, popularised by one of the first IPhOs. Cut off all the resistances which lead out from this bridge, as in Fig. 1. Let x be the equivalent resistance of the part which lies

Figure 1: Ladder: "Cut Out"



to the right (or alternatively, left) of AB. We get, for this part,

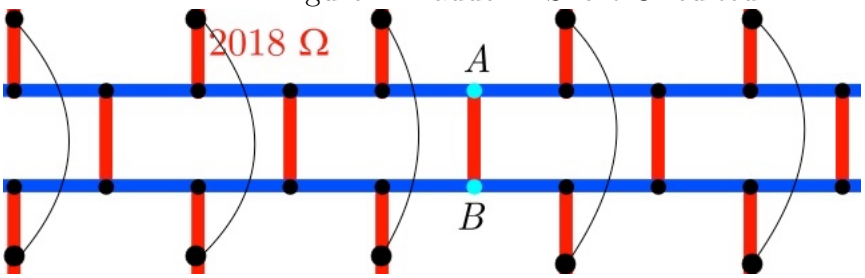
$$\frac{2018\Omega x}{2018\Omega + x} + 4\Omega - x = 0.$$

This may be easily solved; We get, $x = 91.87\Omega$ (Rounding upwards, since we're looking for an upper bound).
For the entire ladder then, we have such section in parallel with a 2018Ω resistor. The equivalent resistance is thus 44.92Ω(rounding up).

Now for the lower bound.
We short circuit each red lead going out from the ladder and lying above

it with the corresponding red lead below the ladder, heading outwards. in other words, letting the black wires have zero resistance, proceed as in Fig. 2. Note that the part outside is now an equipotential; no current flows through

Figure 2: Ladder: "Short Circuited'



it (no potential difference). The part inside is again an infinite ladder, with alternating resistances of 4036Ω and 2018Ω on the 'steps', and 1Ω between any two. Take, again, the portion to the right of AB. Using self-similarity, we get, again,

$$\frac{4036\Omega(\frac{2018\Omega x}{2018\Omega + x} + 2\Omega)}{4036\Omega + (\frac{2018\Omega x}{2018\Omega + x} + 2\Omega)} + 2\Omega - x = 0.$$

Its solution is $x = 74.69\Omega$ (Rounding down); for the actual equivalent resistance, we have, 36.67Ω (Rounding down).[1]
Thus, $\boxed{r = 36.67\Omega}$ and $\boxed{R = 44.92\Omega}$.
Clearly, they satisfy the required conditions.

2 Numerically, Getting Close to The Exact Value

Theoretically, all that'd be required for an exact solution would be making a recursion in 2 variables (for the potential) and solving it. (by adding recursions for (Even, Odd)/(Odd,Even) and (Even,Even)/(Odd,Odd)-type

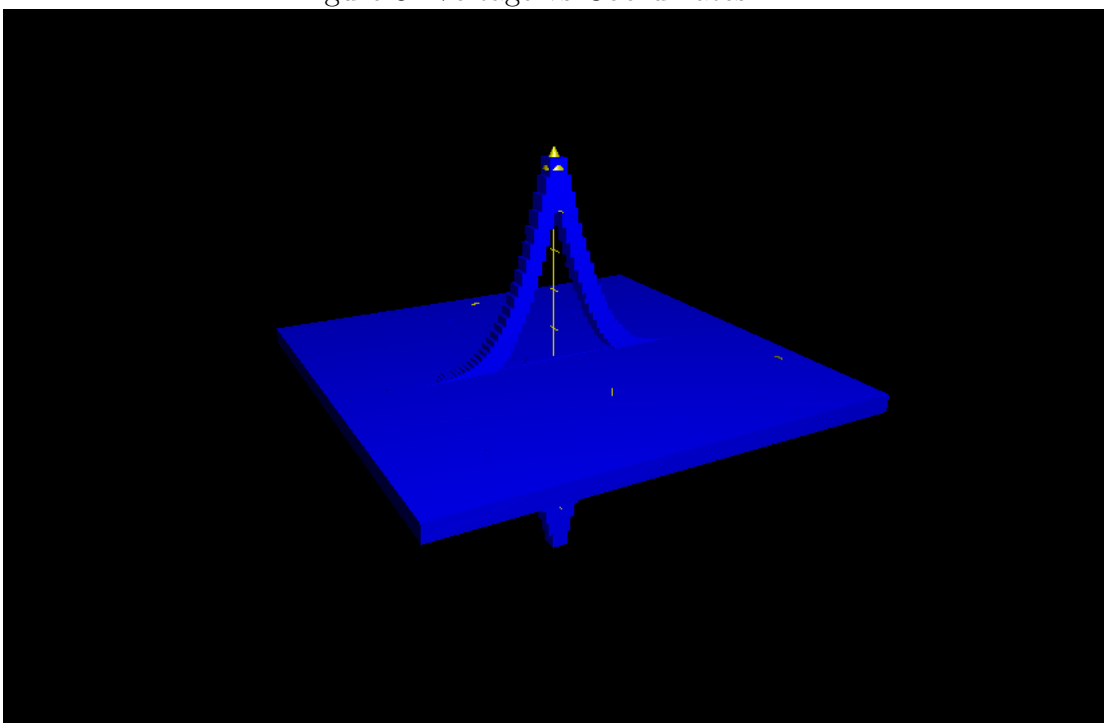
[1]The ugly exact answer may be found here: https://goo.gl/b26fSc

vertices. The first kind have a 2018Ω bridge extending downwards, the second, upwards.[2] However, the analytic solution down such a path seems very daunting. So, I choose to employ numeric means, which are neater, despite the caveat that the computing power is a limitation. In the C++ file attached (code given later, here, too):

1. The grid is rectangular, with vertices at $(0,0)$, (m,n), (n,m), and (n,n).
2. $(\frac{m}{2}, \frac{n}{2})$ is A, and $(\frac{m}{2}, \frac{n}{2}-1)$, B.
3. (i,j) is represented by the matrix element V[i][j].
4. The recursions used for the two kinds of vertices are given under the function step, which operates slightly differently on the two kinds of vertices described above. The recursions are merely "current conservation" equations for the assumed voltage profile.
5. We do something similar to what is done in the relaxation method for attacking Poisson's equation numerically- Each iteration updates values according to the recursions, storing them into the new matrix. Finally, the new one is stored as V.
6. Convergence is slow, but definite.
7. A is kept fixed at $+1V$, and B, at $-1V$.
8. I did preliminary computations for several different grid sizes and iterations ranging in number from 10^4 to 10^5. Convergence seemed slow, and graphs were *qualitatively* the same, as in Figure 3. Some numbers: 700 by 700 grid, (10^5 iterations) and within 20s of units from the centre along the y axis, we see numbers smaller than even 10^{-24}. However, decrease along x-axis is pretty slow; even after a hundred units, we have voltages $\sim 0.01V$, and a hundred more, and just an order of magnitude smaller.
   (Note: From memory; may not be exactly correct) This suggests that using a rectangular grid would be much more efficient; using a 700 by

[2]Note that the nodes lie in a rectangular lattice. We may thus denote their coordinates, taking the unit cell to be one 'brick' from the infinite 'wall'. For an analytic computation, a convenient origin would be A, for a numeric one, we use the lower left vertex as the origin, x-axis to the right, and the y-axis, upwards.

Figure 3: Voltage vs Coordinates



100 grid gave the same results. Also, this makes the computational time scale linearly (instead of the time complexity going as n^2). I could make it twice as fast if I used a 700 by 50 grid, without error, as is evident from the data, but I prefer to err on the side of caution.

9. I've imposed the condition that $V = 0$ at boundaries. Given the potentials at A and B, it seems intuitively obvious.

10. Intuitive reasons behind why the computed value of ρ is likely to be a lower limit:
(a)Bigger grid implies slower drop to zero at large distances (of the potential). Thus, Bigger voltage at nodes neighbouring the centre; lower currents, and higher resistance.
(b)More iterations should result in an increase in potential at nodes, leading to the same thing.

11. The results are more accurate the larger the grid is, and the more

iterations, there are, of course.

12. $$\rho = \frac{2}{2\frac{(V(\frac{m}{2},\frac{n}{2})-V(\frac{m}{2}+1,\frac{n}{2}))}{R_b} + \frac{1-(-1)}{R_r}}$$
. It's obvious, but I include it for the sake of completeness.

13. m and n should be multiples of 4. I could've easily bypassed this, but it uselessly makes the program complicated.

14. The number of iterations is co, $k = \frac{R_r}{R_b}$. For this problem, I've fixed values for R_r and R_b, but that is a trivial change to make.

15. Some values (m,n, number of iterations,R/Ω):
(a) 100,100,100,12
(b)100,100,1000,28
(c)100,100,10000,33
(d)200,200,100,Around 30
(e)400,400,20000,39.55
(f)400,400,40000,39.88
(g)700,700,20000,39.57
(h)700,100,20000,39.57
(i)700,700,100000,40.22
(j)1000,100,100000,40.24 (Again, potential along y-axis dropped very fast (within 30 units) to $10^{-10}V$.)
Better keep n at 100 (faster), and increase co and m.

So, my final estimate for $\boxed{\rho = 40.24\Omega}$, and I've reason to believe that it's slightly lower than the actual value.

16. The program (C++) counts co in 100s, and then prints V at some key points, for a quick analysis and so that one may employ the equation above to find ρ.

17. **The program (C++ version) files the potentials in a file called "Potential". If such a file already exists in that location, it must be deleted to prevent merging of files? Including any file created by a previous instance when this program was run.**

The C++ version:

```
//
//  main.cpp
//  Physics Cup 2018 Pr 1 Improved
//
//  Created by Apple on 02/01/18.
//  Copyright    2018 Siddharth Tiwary. All rights reserved.
//


//Ensure that m and n are multiples of 4
#include <iostream>
#include <vector>
#include <fstream>
#include <string>

std::vector<std::vector<float>> create(int m,int n);
std::vector<std::vector<float>> step(int m,int n,std::vector<std::vector
int main() {
    int m=1000;
    int n=100;
    int cute=m/2;
    int ugly=n/2;
    float k=2018;
    std::vector<std::vector<float>>V=create(m,n);
    for(int co=0;co<100000;co+=1){
        V=step(m,n,V,k);
        if (co%100==0){
            std::cout<<co<<"\n";
        }
    }
    int ice=cute+1;
    int cream=ugly+1;
    std::cout<<V[ice][ugly]<<"\n"<<V[cute][cream]<<"\n"<<V[ice][80]<<"\n
    std::cout<<"Now Writing"<<std::endl;
    std::ofstream outfile;
    outfile.open("Potential", std::ios::app);//Writing potentials to a
    outfile<<"[";
    for(int i=0;i<m;i+=1){
```

```
            outfile<<"]";
            outfile<<"[";
            for(int j=0;j<n;j+=1){
                    std::string s = std::to_string(V[i][j]);
                    outfile<<s;
                    outfile<<" ";

            }
        }
    }
    std::vector<std::vector<float>> create(int m,int n){
        std::vector<std::vector<float>> P;
        for(int i=0;i<m;i+=1){
            std::vector<float> L;
            for(int j=0;j<n;j+=1){
                L.emplace_back(0);
            }
            P.emplace_back(L);
        }
        int cute=m/2;
        int ugly=n/2;
        P[cute][ugly]=1;
        P[cute][ugly-1]=-1;
        return P;
    }
    std::vector<std::vector<float>> step(int m,int n,std::vector<std::vector
        std::vector<std::vector<float>>Q=create(m,n);
        for(int i=1;i<m-1;i+=1){
            for(int j=1;j<n-1;j+=1){
                int cute=m/2;
                int ugly=n/2;
                if (not((i==cute and j==ugly) or (i==cute and j==ugly-1)))
                {
                    if(i%2!=j%2){
                        Q[i][j]=(V[i][j+1]+k*V[i+1][j]+k*V[i-1][j])/(2*k+1)
                    }
                    else{
                        Q[i][j]=(V[i][j-1]+k*V[i+1][j]+k*V[i-1][j])/(2*k+1)
```

```
                }
            }
        }
    }
    return Q;
}
```

They python version is in a more rudimentary form , but for readability:

```
def create(m,n):
    P=[]
    for j in range(0,m):
        L=[]
        for i in range(0,n):
            L.append(float(0))
        P.append(L)
    P[m/2][n/2]=float(1)
    P[m/2][n/2-1]=float(-1)
    return P
def step(m,n,V,k):
    P=create(m,n)
    for i in range(1,m-1):
        for j in range(1,n-1):
            if ((i,j)!=(m/2,n/2) and (i,j)!=(m/2,n/2-1)):
                if (i%2!=j%2):
                    P[i][j]=(V[i][j+1]+k*V[i+1][j]+k*V[i-1][j])/(2*k+1)
                else:
                    P[i][j]=(V[i][j-1]+k*V[i+1][j]+k*V[i-1][j])/(2*k+1)
    return P
m=700
n=100
k=float(2018)
V=create(m,n)
for co in range(1,10000):
    V=step(m,n,V,k)
    if co%100==0:
        print co
```