Cálculo del campo eléctrico a través de la ecuación de Laplace

I, Cely. Y, Porras. J, Melo. A. Vargas.

1. Solución numérica de la situación física

En la figura 1a presenta la afirmación teórica según la cual *"en el interior de un conductor el campo eléctrico es nulo"*, de acuerdo con la expresión que relaciona la diferencial de potencial eléctrico y el campo:

$$\vec{E} = -\vec{\nabla}V$$

es preciso afirmar que si la diferencia de potencial es cero (como en el caso del interior del conductor) el campo eléctrico es nulo.

2. Eficiencia de los métodos de relajación

Los métodos usados fueron los siguientes:

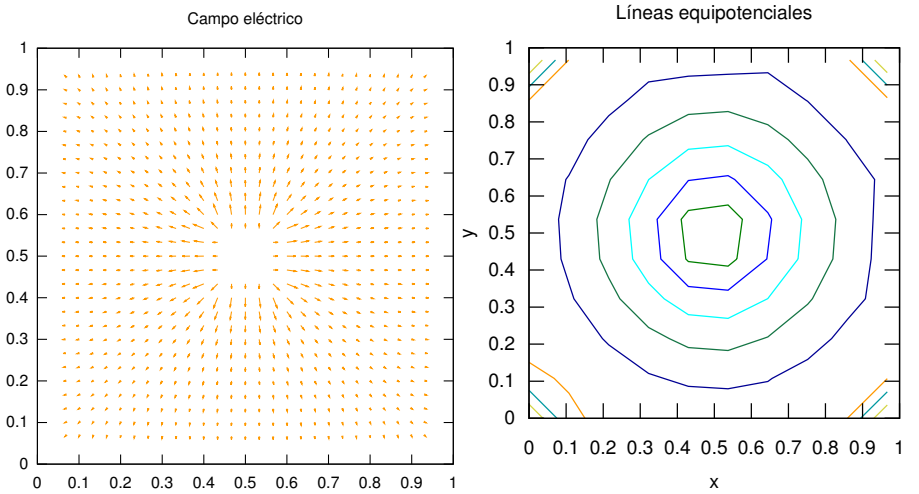
- Método de Jacobi
- Método Gaussian-Seidel (gsei)
- Método Gaussian-Seidel with Succesive Over-Relaxation (sor)

Una forma de medir la eficiencia de cada método consiste en establecer básicamente dos relaciones, la primera es qué tantas iteraciones requiere el método para determinada cántidad de cifras decimales significativas, la otra qué tantas iteraciones requiere para determinada dimensión de la red. En la figura 5 se presentan los resultados de las mediciones realizadas, la figura 5a muestra que el método "Gaussian-Seidel with Succesive Over-Relaxation (sor)" es el más eficiente de los tres algoritmos utilizados para resolver el problema ya que presenta la menor pendiente es decir para obtener un mayor nivel de precisión se requiere el menor número de iteraciones comparado con los métodos de Jacobi y "Gaussian-Seidel (gsei)" ésta medición se realizó para la configuración geométrica mostrada en la figura 2c variando unicamente el factor de convergencia, igualmente la comparación de la eficiencia entre los métodos de Jacobi y SOR para la dimensión de red (figura 5b) evidencia que la solución converge rápidamente mediante SOR, sin embargo, no se logró establecer la proporcionalidad debido a que el ajuste realizado no proporciona resultados satisfactorios, ésta última medición se realizó variando la variable N que establece la dimensión de la red cuadrada ($N * N$) con una valor del factor de convergencia fijado a $1 * 10^{-10}$ en la geometría mostrada en la figura 2c. Por medio de tanteo se obtuvo que el valor más eficiente del parámetro de optimización (o sobre-relajación) para SOR es $\omega = 1{,}69$ el cual fue utilizado en las mediciones mencionadas anteriormente.

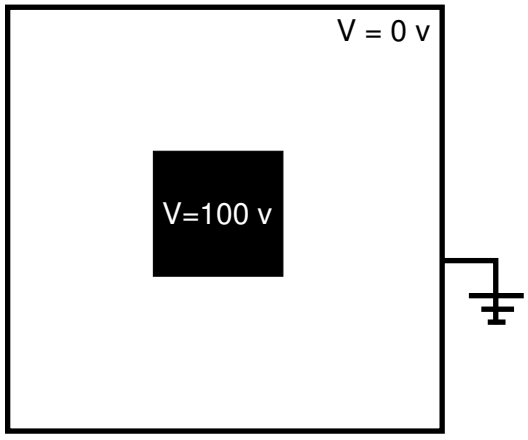
3. Código en C++

```
/*
  This program was created in order to
  study the electric field generated by
  different geometries.

  sintaxis:
  $sor x method
```

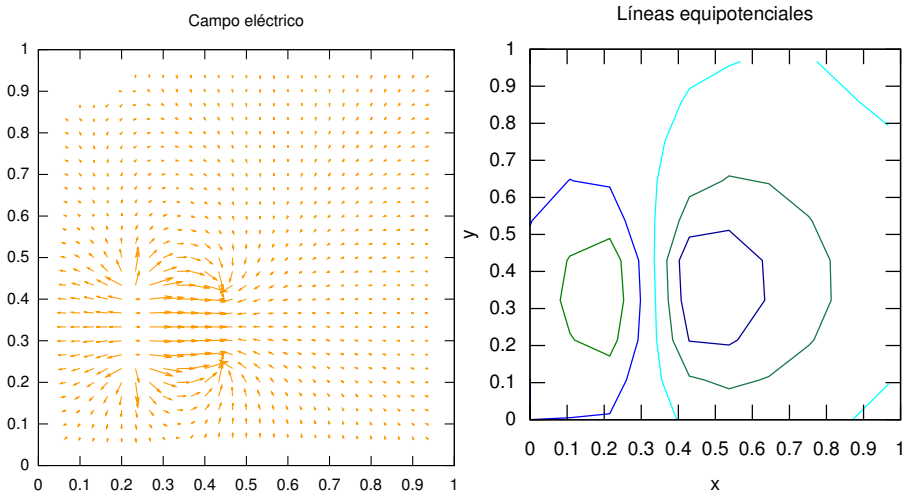


(a) Configuración del campo eléctrico (b) Configuración de las líneas equipotenciales

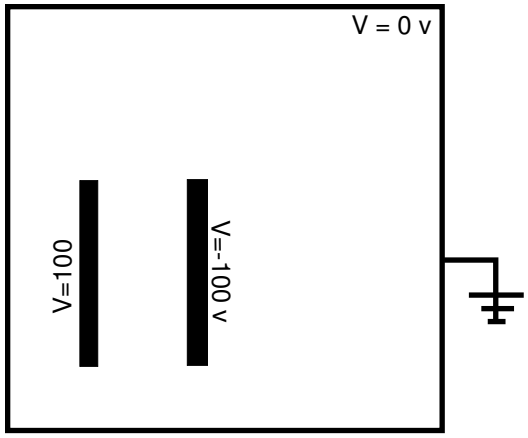


(c)

Fig. 1: Sección transversal de un conductor sólido mantenido a V =100 V dentro de una caja metálica hueca (a V =0 V).

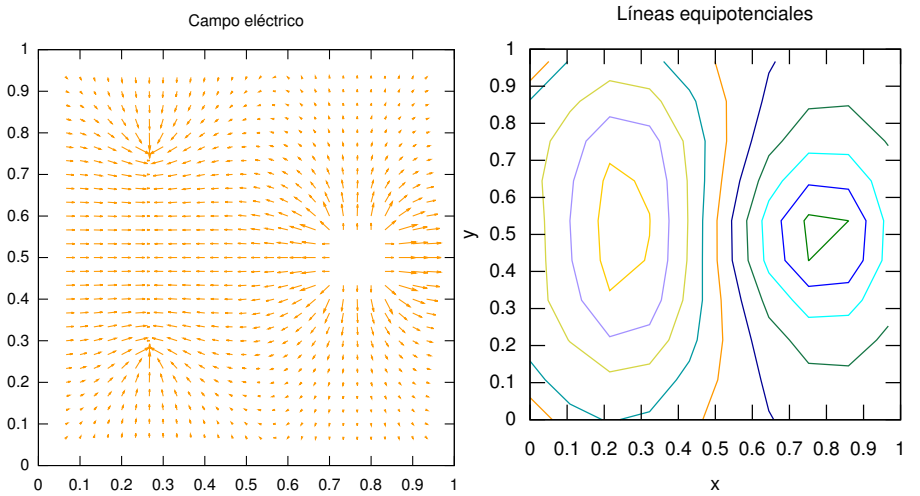


(a) Configuración del campo eléctrico (b) Configuración de las líneas equipotenciales

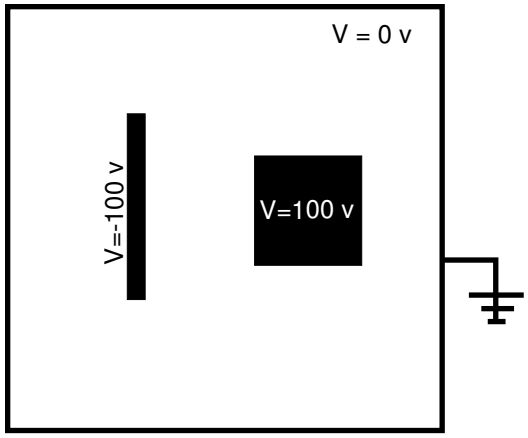


(c)

Fig. 2: Capacitor de placas paralelas: una placa a +100 V y la otra a -100 V, encerrado dentro de una caja metálica a V =0 V.

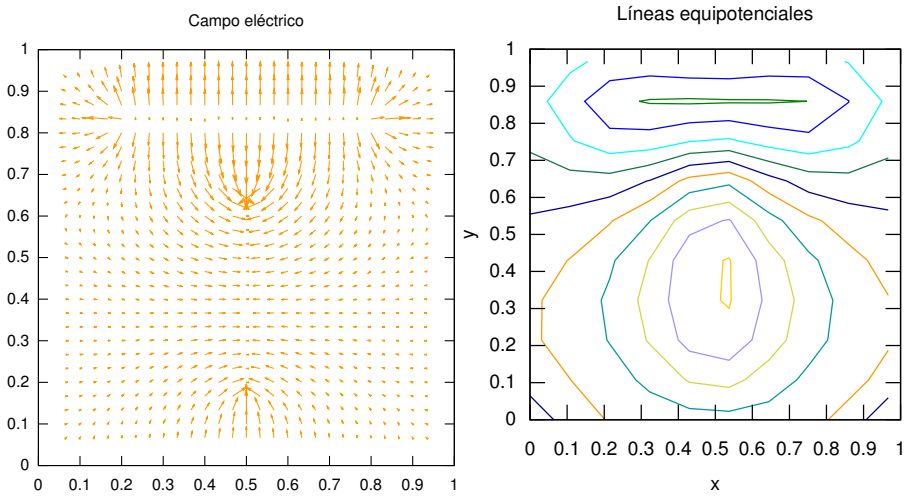


(a) Configuración del campo eléctrico (b) Configuración de las líneas equipotenciales

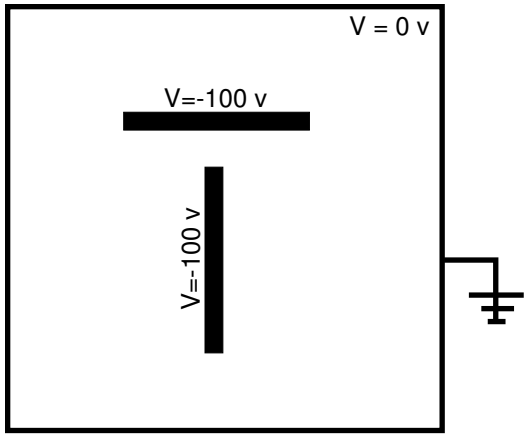


(c)

Fig. 3: Sistema placa metálica a V=-100 a V y conductor metálico mantenido a +100 V a encerrados dentro de una caja a V=0 V.

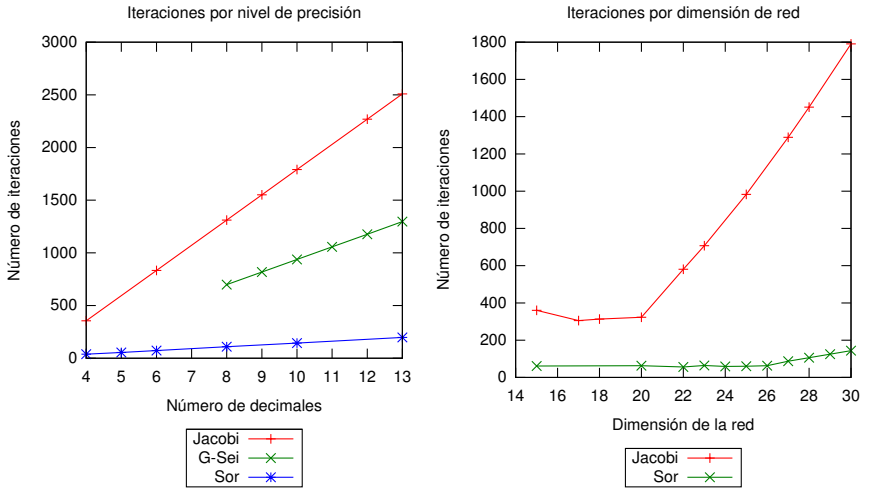


(a) Configuración del campo eléctrico (b) Configuración de las líneas equipotenciales



(c)

Fig. 4: Modelo de un pararrayos. Las dos placas a +100 V y -100 V estan encerradas dentro de una caja metálica a V=0 V.



(a) Número de iteraciones por nivel de precisión

(b) Número de iteraciones por dimensión de la red

Fig. 5: Comparación de la eficiencia de los métodos utilizados

```
  where x is the number of the figure
  only from 0 to 4 and method represents
  the numerical method to apply:
  gesi (Gaussian-Seidel)
  jacobi (Jacobi)
  sor (Gaussian-Seidel with Successive Over-Relaxation)
*/

#include <iostream>
#include <math.h> //only for fabs's method
#include <fstream>
#include <stdlib.h> //only for system method
#include <string.h>
using namespace std;

#define omega 1.695//¿?
#define N 30

const int maxiter = 5e3;//Iteration's number
int i = N, j= N;
double tol = 1.0e-13; //Convergence's canon
double dmax = 0.0;
double tmp,r,x,y,dx;
double v [N][N], Ex [N][N],Ey [N][N];
double vold[N][N];
bool geo[N][N]; //object to define the geometry only once time
fstream infile;

void draw_geometry(){
```

```
  //Can I draw the geometry before?
  if (geo[i][j]==false) {
    if (j==0||j==N-1||i==0||i==N-1) {
      //to mass
      cout << "0 ";
    }
    else {
      //point
      cout << "· ";
    }
  }
  else{
    //geometry
    cout << "1 ";
  }
}

void sor () {
  double vold;
  int n;
  for (n=1; n < maxiter ; n++)
  {
    dmax = 0;
    for(i = 2; i < N-1; i++)
    {
      for(j = 2; j < N-1 ;j++)
      {
        //Geometry's exclussion
        if (geo[i][j]==true)
        {
          cout << "Geometry's point, skiped!\r";
        }
        else
        {
          cout << "I'm working, Don't worry!\r";
          vold = v[i][j];
          tmp=0.25*(v[i+1][j]+v[i-1][j]+\
          v[i][j+1]+v[i][j-1]); //Valor temporal de V
          r=omega*(tmp-v[i][j]); //valor de r donde v[i][j] es Vold
          v[i][j]=v[i][j]+r; //valor de Vnew
          dmax = fabs(v[i][j]-vold);
          if(r > dmax) dmax=r;

        }
      }
    }
    if(dmax < tol)
    {
      cout << "Tolerance's margin arrived at.\t\t[Ok]\n";
      break;
    }
```

```
  }
  cout << "The value of tolerance is = " << tol <<endl;
  cout << "I did "<< n <<" iterations.\n";
}

void jacobi (){
  int n;
  for (n=1; n < maxiter ; n++)
  {
    //I'm going to find a new grid
    dmax = 1;
    for(i = 2; i < N-1; i++)
    {
      for(j = 2; j < N-1 ;j++)
      {
        //Geometry's exclussion
        if (geo[i][j]==true)
        {
          cout << "Geometry's point, skiped!\r";
        }
        else
        {
          cout << "I'm working, Don't worry!\
          \t\t["<< n << "]\r" ;
          v[i][j]=0.25*(vold[i+1][j]+vold[i-1][j]+\
          vold[i][j+1]+vold[i][j-1]); //Valor temporal de V
          if (n>100) dmax = fabs(v[i][j]-vold[i][j]);
          //cout << dmax << endl;
        }
      }
    }

    //I specify again a new old grid
    for(i = 2; i < N-1; i++)
    {
      for(j = 2; j < N-1 ;j++)
      {
        //Geometry's exclussion
        if (geo[i][j]==true)
        {
          cout << "Geometry's point, skiped!\r";
        }
        else
        {
          cout << "I'm working, Don't worry!\r"  ;
          vold[i][j]=v[i][j]; //Valor temporal de V

        }
      }
    }

    if(dmax < tol)
```

```
    {
      cout << "Tolerance's margin arrived at.\t\t[Ok]\n";
      break;
    }

  }
  cout << "The value of tolerance is = " << tol <<endl;
  cout << "I did "<< n <<" iterations.\n";
}


void gsei (){
  double vold;
  int n;
  for (n=1; n < maxiter ; n++)
  {
    dmax = 0;
    for(i = 2; i < N-1; i++)
    {
      for(j = 2; j < N-1 ;j++)
      {
        //Geometry's exclussion
        if (geo[i][j]==true)
        {
          cout << "Geometry's point, skiped!\r";
        }
        else
        {
          vold = v[i][j];
          cout << "I'm working, Don't worry!\r";
          v[i][j]=0.25*(v[i+1][j]+v[i-1][j]\
          +v[i][j+1]+v[i][j-1]); //Valor temporal de V
          dmax = fabs(v[i][j]-vold);
        }
      }
    }
    if(dmax < tol)
    {
      cout << "Tolerance's margin arrived at.\t\t[Ok]\n";
      break;
    }

  }
  cout << "The value of tolerance is = " << tol <<endl;
  cout << "I did "<< n <<" iterations.\n";
}


void electric_field(){
  //Electric field
  double fac=5.0e-5; // scaling factor for vector plot
  dx=1.0/N; //finite difference
```

```
  for(i=2; i < N-1; i++)
  { //To iterate along y
    for(j=2; j < N-1; j++)
    { //To iterate along x
      //is unneeded to exclude the geometry here
      cout << "I'm working on the electric field.\r";
      Ex[i][j]=-(v[i+1][j]-v[i-1][j])*(1/(2*dx))*fac;
      Ey[i][j]=-(v[i][j+1]-v[i][j-1])*(1/(2*dx))*fac;
    }
  }
  cout << "Electric field calculated.\t\t[OK]\n";

  //to fstream object
  infile << "#x y v Ex Ey\n";
  for(i=0; i < N; i++)
  {
    x=i*dx;
    for(j=0; j < N; j++)
    {
      y=j*dx;
      infile << x << "\t" << y << "\t" << v[i][j] << "\t";
      infile << Ex[i][j] << "\t" << Ey[i][j] << "\n";
    }
  }
  cout << "\nThe file \"data.dat\" was generated,\n";

  infile.close();

  //system("gnuplot gplot.gp"); //first gnuplot's output is wrong
}

int main(int argc,char* argv[]) {

  system(">data.dat");
  infile.open("data.dat");

  cout << "The geometry "<< argv[1] <<" looks like:\n\n\t";
  for (j = N-1; j >= 0 ;j--)
  {
    for (i = 0; i < N;i++)
    {
      //Geometry's definition
      if (strcmp(argv[1],"0")==0){
        if (floor((i-15)*(i-15)+(j-15)*(j-15))>=90 &&\
        floor((i-15)*(i-15)+(j-15)*(j-15))<=140){
          vold[i][j]=v[i][j]=50;
          geo[i][j] = true;
        }
        else
        {
          geo[i][j] = false;
          vold[i][j]=v[i][j] = 0.0;
```

```
    Ex[i][j] = 0.0;
    Ey[i][j] = 0.0;
  }
}
//~~~~~~figure~1~~~~~~~~~~~~
if (strcmp(argv[1],"1")==0){

  if ((i>=13)&&(i<=17)&&(j>=13)&&(j<=17)){
    vold[i][j]=v[i][j] = 100.0;
    geo[i][j] = true;
  }
  else
  {
    geo[i][j] = false;
    vold[i][j]=v[i][j] = 0.0;
    Ex[i][j] = 0.0;
    Ey[i][j] = 0.0;
  }
}
//~~~~~~figure~2~~~~~~~~~~~~
if (strcmp(argv[1],"2")==0){
  if ((i==7)&&(j>=7)&&(j<=13))
  {
    vold[i][j]=v[i][j] = 100.0;
    geo[i][j] = true;
  }
  else if ((i==13)&&(j>=7)&&(j<=13))
  {
    vold[i][j]=v[i][j] = -100.0;
    geo[i][j] = true;
  }
  else
  {
    geo[i][j] = false;
    vold[i][j]=v[i][j] = 0.0;
    Ex[i][j] = 0.0;
    Ey[i][j] = 0.0;
  }
}
//~~~~~~figure~3~~~~~~~~~~~~
if (strcmp(argv[1],"3")==0){

  if ((i>=21)&&(i<=25)&&(j>=13)&&(j<=17)){
    vold[i][j]=v[i][j] = 100.0;
    geo[i][j] = true;
  }
  else if ((i==8)&&(j>=8)&&(j<=23)){
    vold[i][j]=v[i][j] = -100.0;
    geo[i][j] = true;
  }
  else
  {
```

```
          geo[i][j] = false;
          vold[i][j]=v[i][j] = 0.0;
          Ex[i][j] = 0.0;
          Ey[i][j] = 0.0;
        }
      }
      //~~~~~~figure~4~~~~~~~~~~~~
      if (strcmp(argv[1],"4")==0){

        if ((j==25)&&(i>=6)&&(i<=24)){
          vold[i][j]=v[i][j]=100;
          geo[i][j] = true;
        }
        else if ((i==15)&&(j>=5)&(j<=20)){
          vold[i][j]=v[i][j]=-100;
          geo[i][j] = true;
        }
        else
        {
        geo[i][j] = false;
        v[i][j] = 0.0;
        Ex[i][j] = 0.0;
        Ey[i][j] = 0.0;
        }
      }

      draw_geometry();
    }
    i=0;
    cout << endl << "\t";
  }

  cout << "\nInit variables,\t\t\t\t[Ok]\n";

  if (strcmp(argv[2],"jacobi")==0){
    jacobi();
  }
  else if (strcmp(argv[2],"gsei")==0){
    gsei();
  }
  else if (strcmp(argv[2],"sor")==0){
    sor();
  }
  electric_field();

  return 0;



}
```