Determinación de la densidad de carga de un conductor por métodos numéricos

Paul Rosa Ruiz y José Ortega Moya

Febrero 2021

Resumen

En este documento se abordará el problema del cálculo de la distribución superficial de carga en un conductor de forma que sea equipotencial. Para ello se llegará a una ecuación que describe la densidad de carga en un punto del conductor en función de la geometría y densidad de carga en el resto de la superficie de este. Y, mediante un proceso numérico iterativo se llegará a una solución autoconsistente.

Palabras clave: Densidad de carga, conductor, potencial, campo eléctrico, método numérico.

1. Introducción

El cálculo de la distribución superficial de carga sobre un conductor no es en general, un problema trivial puesto que este debe ser equipotencial y la carga se debe distribuir por la superficie de este cumpliendo esta condición.

Para buscar una expresión que nos permita calcular la densidad superficial de carga en la superficie metálica se parte de que en el interior de este el campo es nulo y que la condición de frontera entre medios viene dada por:

$$D_{2n} - D_{1n} = \sigma_f \tag{1}$$

Donde D_{2n} es el campo vector Desplazamiento fuera del conductor y en dirección normal a su superficie y $D_{1n} = 0$ en el interior. De esta forma el campo en dirección normal a la superficie en puntos cercanos a esta viene dado por:

$$E_n = \frac{\sigma}{\varepsilon_0} \tag{2}$$

Si consideramos que la superficie está formada por discos infinitesimales con una densidad superficial de carga σ, cada uno de ellos crea un campo:

$$E = \frac{\sigma}{2\varepsilon_0}(1 - \cos\alpha) \tag{3}$$

Y para puntos infinitamente cerca de la superficie $\alpha \approx \pi/2$, luego:

$$E = \frac{\sigma}{2\varepsilon_0} \tag{4}$$

De esta forma obtenemos que, dado que el campo total es la suma de las contribuciones del disco y del resto de la superficie, esta última debe ser $\frac{\sigma}{2\varepsilon_0}$.

Dado que el disco no crea campo sobre sí mismo, el campo en la superficie debe ser precisamente:

$$E = \frac{\sigma}{2\varepsilon_0} \tag{5}$$

Así:

$$\vec{E} \cdot \hat{n} = \int\int_{S'} \frac{1}{4\pi\varepsilon_0} \frac{\sigma' \hat{r} \cdot \hat{n}}{r^2} dS' = \frac{\sigma}{2\varepsilon_0} \tag{6}$$

De donde se obtiene la ecuación:

$$\sigma(x,y,z) = \frac{1}{2\pi} \int\int_{S'} \frac{\sigma'(x',y',z')\cos(\theta)}{r^2} dS' \tag{7}$$

Por otra parte, si existe campo externo se obtiene un resultado análogo:

$$\sigma(x,y,z) = 2\varepsilon_0(\vec{E} \cdot \hat{n}) + \frac{1}{2\pi} \int\int_{S'} \frac{\sigma'(x',y',z')\cos(\theta)}{r^2} dS' \tag{8}$$

2. Métodos

Para el tratamiento numérico del problema en general, lo primero es necesario encontrar una parametrización $\mathbf{S}$ de la superficie que se esté estudiando:

$$\mathbf{S}(u,v) = (x,y,z) \tag{9}$$

donde $(u,v) \in U \times V \subset \mathbb{R}^2$ con U, V intervalos.

Una vez hecho esto se puede proceder a la integración numética sobre la superficie, lo cual se hará discretizando los intervalos U, V en divisiones lo sufiecientemente pequeñas como para obtener:

$$\int\int_{S'} \frac{\sigma'(x',y',z')\cos(\theta)}{r^2} dS' = \int_{u'}\int_{v'} \frac{\sigma'(\mathbf{S}(u',v'))\cos(\theta)}{||\mathbf{S}(u,v)-\mathbf{S}(u',v')||^2} \left|\left|\frac{\partial \mathbf{S}}{\partial u}\times\frac{\partial \mathbf{S}}{\partial v}\right|\right| dudv \approx \sum_{u'}\sum_{v'} \frac{\sigma'(\mathbf{S}(u',v'))\cos(\theta)}{||\mathbf{S}(u,v)-\mathbf{S}(u',v')||^2} \left|\left|\frac{\partial \mathbf{S}}{\partial u}\times\frac{\partial \mathbf{S}}{\partial v}\right|\right| \Delta u\Delta v \quad (10)$$

dónde $\frac{\partial \mathbf{S}}{\partial v} = \left(\frac{\partial S_x}{\partial v}, \frac{\partial S_y}{\partial v}, \frac{\partial S_z}{\partial v}\right)$ y $\frac{\partial \mathbf{S}}{\partial u} = \left(\frac{\partial S_x}{\partial u}, \frac{\partial S_y}{\partial u}, \frac{\partial S_z}{\partial u}\right)$, son vectores tangentes a la superficie en los puntos (u,v) y se calcularán analíticamente para poder evaluarlos en el programa. Por otra parte, su producto vectorial es un vector normal a la superficie y $\left|\left|\frac{\partial \mathbf{S}}{\partial u}\times\frac{\partial \mathbf{S}}{\partial v}\right|\right| \Delta u\Delta v$ es el módulo del diferencial de superficie $\Delta S'$.

Para hallar $cos(\theta)$, el ángulo entre los vectores normal a la superficie en (x,y,z): $\vec{n} = \left(\frac{\partial \mathbf{S}}{\partial u}\times\frac{\partial \mathbf{S}}{\partial v}\right)$ y $\vec{r} = (x-x', y-y', z-z')$ se empleará:

$$\cos(\theta) = \frac{\vec{n}\cdot\vec{r}}{||\vec{n}||\,||\vec{r}||} \quad (11)$$

Mediante este proceso sumatorio se puede evaluar la expresión 7 para obtener un nuevo valor para la densidad de carga en cada punto $\sigma(x,y,z)$ de la superficie teniendo en cuenta los antiguos valores $\sigma'(x',y',z')$ en el resto de la superficie. De esta forma, repitiendo este proceso de forma iterada con los nuevos valores obtenidos, el resultado debería aproximarse cada vez más a una solución concreta. La solución se considerará correcta cuando la media de variaciones en la densidad de carga en los puntos sea menor que una determinada precisión ϵ.

A la hora de hacer el sumatorio surgen varios problemas. En primer lugar es que no puede darse $(u,v) = (u',v')$ pues entonces el denominador en la expresión 10 se anularía, dando un resultado indefinido, esto tiene sentido, pues un diferencial de carga no crea campo sobre sí mismo. Esto se arregla añadiendo un condicional al bucle que realiza la integral de forma que si $(u,v) = (u',v')$, entonces no lo tenga en cuenta. El segundo problema tiene que ver con la inyectividad de la parametrización. Si no lo es tendremos que existirán al menos don puntos $(u',v'),(u,v)$ cumpliendo $(u,v) \neq (u',v')$ tales que $\mathbf{S}(u,v) = \mathbf{S}(u',v')$, llevándonos al mismo error anterior. Este caso deberá ser solucionado para cada geometría en particular.

Para considerar el caso en el que existe campo externo simplemente se realiza el mismo proceso pero empleando la expresión 8.

2.1. Funciones del programa

Para llevar a cabo el proyecto, se crearon varias funciones que optimizaran el código y realizaran tareas repetitivas más fácilmente (por ejemplo, productos vectoriales, productos escalares, etc.).

conversionIndividual() Es una función para proporcionalidades

cCoord() Dadas dos variables y el tipo de geometría (esfera o toroide) devuelve el punto de $\mathbb{R}^3$ del punto de la superficie (equivalente a las funciones descritas en las Ecuaciones 12 y 13.

prodEscalar() Devuelve el producto escalar entre dos vectores. Necesario ya que en la Ecuación de Robin hay un producto escalar en la integración.

norma() Devuelve la norma de un vector

coseno() Devuelve el coseno entre dos vectores

prodVectorial() Devuelve el producto vectorial entre dos vectores. Útil para calcular los vectores normales a la superficie en cada punto.

vectDiferencia() Da como resultado el vector diferencia entre dos puntos definidas en un espacio Euclídeo estándar.

compara() Compara dos valores y devuelve si su diferencia es mayor o menor a la precisión introducida

definirXYZ() Crea tres matrices X, Y y Z bidimensionales, los cuales tienen guardados las coordenadas x, y, z, respectivamente, en para cada (u,v). Es decir, $X[u,v] = x(u,v)$.

crearVectorOrtogonal() Crea una matriz VectorOrtogonal y otra NormadS con las componentes del vector ortogonal a la superficie de cada punto y la norma de esta (el área del diferencial de área), respectivamente.

imprimirFichero() Dada una matriz con los valores de la densidad de carga para cada punto (u,v), lo escribe en un fichero para, luego, poder leerlo desde Matlab y representarlo gráficamente para ver el resultado.

integral() Para cada punto (u, v) integra la expresión de la Ecuación de Robin. (diferente para cada geometría).

2.2. Casos particulares

2.2.1. Esfera

En primer lugar es necesario encontrar una parametrización de la superficie esférica:

$$\mathbf{S}(u,v) = (x,y,z) \quad \text{con} : \begin{cases} x = R\cos(u)\sin(v) \\ y = R\sin(u)\sin(v) \\ z = R\cos(v) \end{cases} \tag{12}$$

dónde $(u, v) \in [0, \pi) \times (0, \pi)$ y R es el radio de la esfera.

Se observa que esta parametrización supone un problema de inicio pues los puntos situados en los polos ($z = \pm R$) no están incluidos al corresponderse con $\cos(v) = 1$, que se da con $v = 0$ y $v = \pi/2$. Y si se trata de incluirlos haciendo $v \in [0, \pi/2]$ se llega a una parametrización no inyectiva, es decir, cualquier punto $(u, v) = (u, 0)$ o $(u, v) = (u, \pi)$ tendría la misma imagen, lo que nos llevaría a contar varias veces el mismo punto y a errores como que la distancia entre puntos que deberían ser distintos es nula.

Para buscar los vectores normales a la superficie en cada punto se emplea el producto vectorial de las derivadas de $\mathbf{S}$ frente a los parámetros (u, v). En este caso las derivadas son:

$$\frac{\partial \mathbf{S}}{\partial u} = \left(\frac{\partial S_x}{\partial u}, \frac{\partial S_y}{\partial u}, \frac{\partial S_z}{\partial u}\right)$$

$$\begin{cases} \frac{\partial S_x}{\partial u} = -R\sin(v)\sin(u) \\ \frac{\partial S_y}{\partial u} = R\cos(u)\sin(v) \\ \frac{\partial S_z}{\partial u} = 0 \end{cases}$$

$$\frac{\partial \mathbf{S}}{\partial v} = \left(\frac{\partial S_x}{\partial v}, \frac{\partial S_y}{\partial v}, \frac{\partial S_z}{\partial v}\right)$$

$$\begin{cases} \frac{\partial S_x}{\partial v} = R\cos(v)\cos(u) \\ \frac{\partial S_y}{\partial v} = R\sin(u)\cos(v) \\ \frac{\partial S_z}{\partial v} = -R\sin(v) \end{cases}$$

El problema de la inyectividad se tratará de resolver con una serie de condicionales tipo `if(...){ ...} else{...}` en el programa, de forma que solo se cuente una vez los puntos de cada polo: $(u, v) = (0, 0)$ y $(u, v) = (0, \pi)$. Por lo tanto al integrar se ignorarán los puntos $(u', v') = (u', 0)$ y $(u', v') = (u', \pi)$ con $u' \neq 0$. Así tenemos varios casos según el punto (u, v) sobre el que se va a calcular la nueva densidad de carga:

- Si $v = 0$ o $v = \pi$ y $u = 0$ entonces estos serán los puntos que sí contaremos para los polos, por lo que se hace la integral.
- Si $v \neq 0$ y $v \neq \pi$ se realiza la integral.
- Si $v = 0$ o $v = \pi$ y $u \neq 0$ Entonces se le asigna el mismo valor que al punto $(0, v)$ ya que para la representación gráfica es necesario que tenga ese valor, aunque para los cálculos lo ignoremos.

2.2.2. Toroide

En el caso del Toro, el primer paso trata de parametrizar la superficie de manera que con dos variables cualquier punto del toro está localizado. En este caso, la Ecuación 13 describe la parametrización usada para este trabajo.

$$f : [0, 2\pi) \times [0, 2\pi) \to \mathbb{R}^3 \tag{13}$$

$$(u, v) \mapsto ((R + r\cos(v))\cos(u), (R + r\cos(v))\sin(u), r\sin(v))$$

Donde R es el radio del toroide y r el radio del cilindro (ver Figura 1).

Esta parametrización está bien definida ya que la función f es inyectiva y es un difeomorfismo. En otras palabras, $\forall(u, v) \quad \exists!(x, y, z) \in \mathbb{R}^3$ tal que $f(u, v) = (x, y, z)$.

Igual que se hizo con la esfera, los vectores normales a la superficie del toroide vienen definidos por las derivadas parciales de f respecto a u y v.

$$\frac{\partial f}{\partial u} \times \frac{\partial f}{\partial v} = (n_x, n_y, n_z) \tag{14}$$

Para calcular los vectores normales, la función crearVectorOrtogonal invoca a prodVectorial y así obtener todos los vectores normales de cada punto de la superficie del Toroide

```
1 void crearVectorOrtogonal(int tipo)
2 void prodVectorial(double ux,double uy,
3 double uz, double vx, double vy, double vz,
4 double & wx, double & wy, double & wz)
```

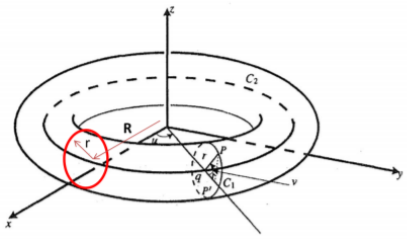


Figura 1: Toroide

3. Resultados

3.1. Esfera

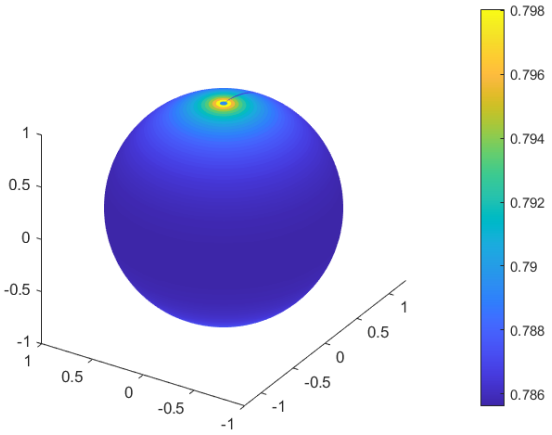
A pesar de los intentos realizados, se obtienen resultados algo incoherentes para el caso de la esfera cargada, con densidades de carga ligeramente superiores en los polos, cuando, por simetría, debería ser igual en todas partes.

Veamos un caso como ejemplo. Se toma una esfera de radio unidad con una carga total $Q = 10C$ y campo externo nulo y se discretizan los intervalos U, V en 100 divisiones cada uno, dando lugar a 10000 divisiones de la superficie. La teoría nos dice que la densidad de carga debe ser uniforme $\sigma = Q/S = 10/(4\pi) = 0{,}796C/m^2$. Si observamos la figura 2a, donde se plasman los resultados obtenidos, se ve que para cada punto la densidad se encuentra entre $0,798$ y $0,786C/m^2$, suponiendo un error máximo del 1,3 %. Sin embargo, aunque el error no sea excesivo, el hecho de que la carga no se distribuya uniformemente es indicador de que algo se está haciendo mal pues, en teoría las operaciones que se hacen en cada punto son iguales. Hay tres opciones principales para explicar esto:

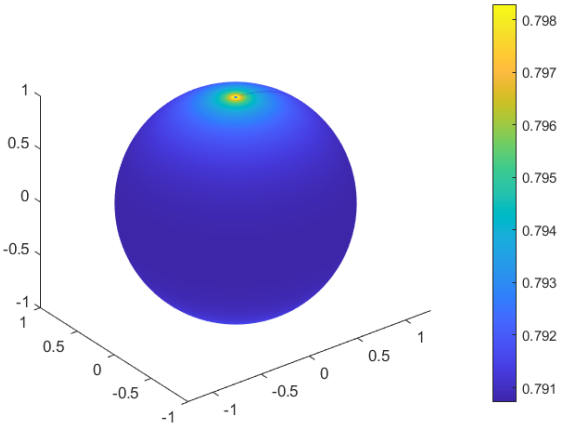
- Error en la programación.
- Error en la en la elección del método empleado.
- Distribución heterogénea del los errores numéricos en la integración.

La hipótesis del error numérico parece tomar algo de importancia si se repite la operación pero dividiendo los intervalos en 200 (figura 2b). El error máximo se reduce hasta un 0,63 % y las anomalías en los polos se concentran en un espacio más pequeño.

Cuando se trata una esfera de iguales características, pero con carga nula e inmersa en un campo $\vec{E} = 10kV/m \quad \hat{x}$ se obtienen resultados coherentes (figura 3),



(a) 10000 divisiones

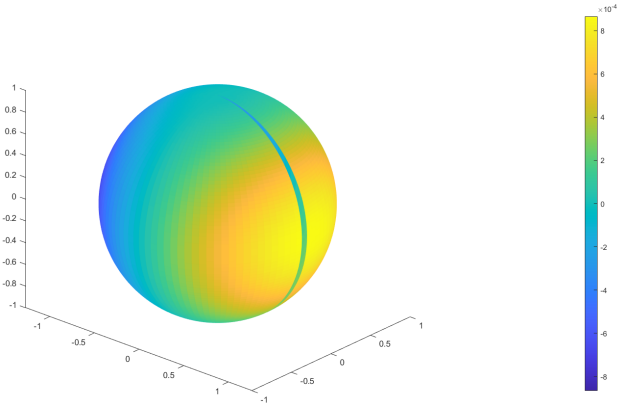


(b) 40000 divisiones

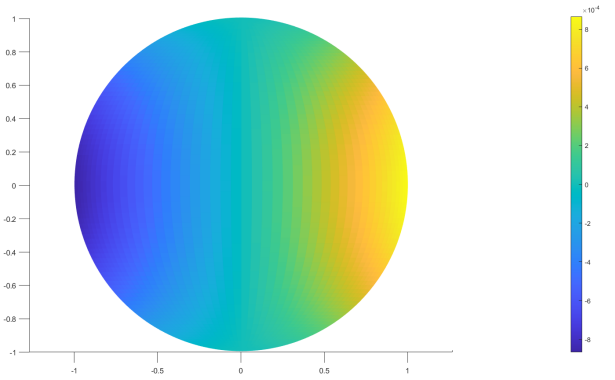
Figura 2: Esfera sin campo externo y con una carga de 10C.

con un desplazamiento de la carga positiva hacia valores positivos del eje x y la carga negativa hacia valores negativos del eje x.

En el caso de combinar una esfera con carga $1C$ y un campo $\vec{E} = 10kV/m \quad \hat{x}$, vuelven a aparecer los errores anteriores (Figura 4)



(a) Vista 1



(b) Vista 2

Figura 3: Esfera sin carga y con un campo de +10kV/m en dirección x.

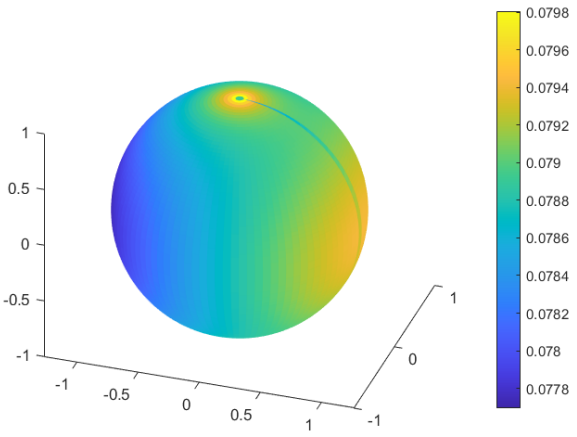


Figura 4: Esfera con una carga de 1C y un campo de +10kV/m en dirección x.

toroide con 1C.

3.2. Toroide

Los resultados obtenidos para el Toroide son, en nuestra opinión, coherentes. Como se esperaba, las zonas con más densidad de carga son las externas ya que son los puntos más alejados entre ellos y las cargas se repelen. La Figura 5 nos muestra la distribución de las densidades de carga obtenidas para un conductor aislado con radio mayor 3m y radio menor 1m con una carga de 10C. Esta representación ha sido obtenido por Matlab tras leer el archivo generado por el código (concretamente la función imprimirFichero()), el cual contiene la cantidad de densidad de carga para cada punto determinada por la parametrización (ver Ecuación 13).

En la figura 6 se representa el mismo toroide, esta vez sin carga y con un campo de $\vec{E} = 10kV/m \quad \hat{x}$ y en la figura 7 se mantiene el mismo campo y se carga el

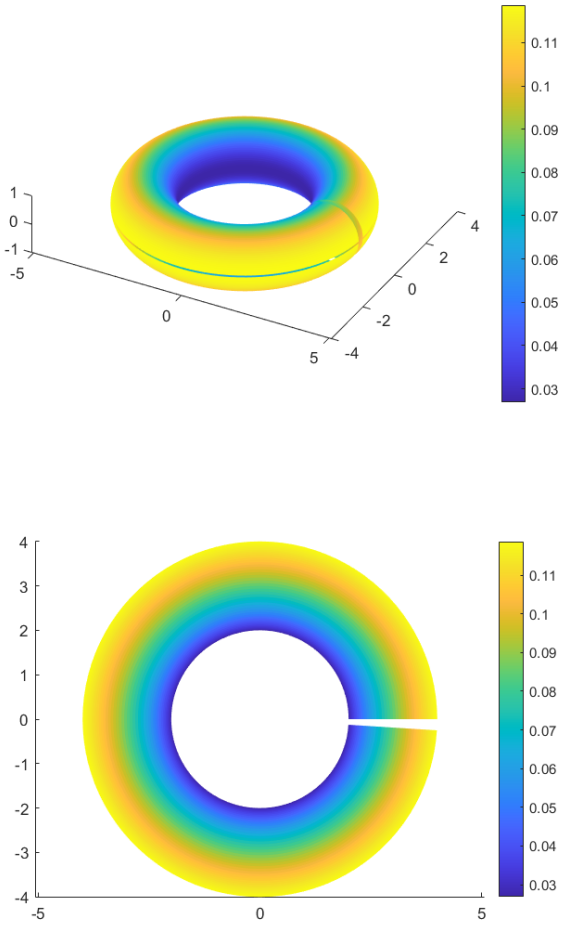


Figura 5: Distribución de un conductor con forma de toroide aislado con una carga de 10C.

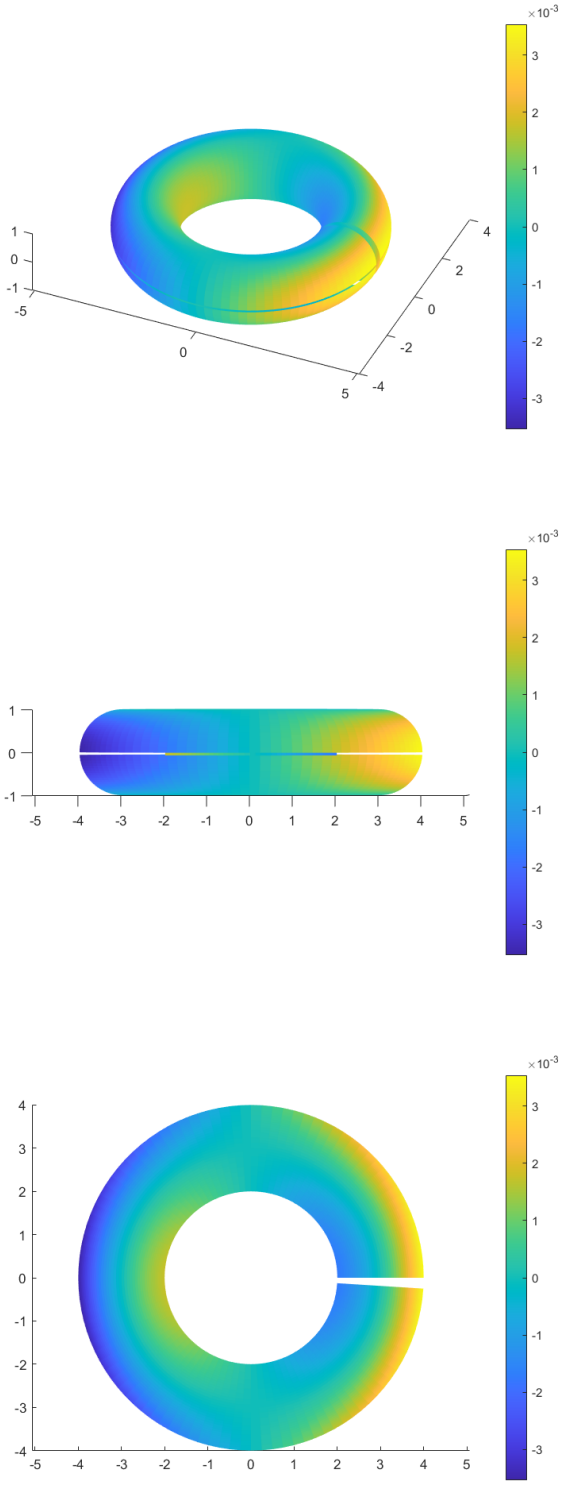


Figura 6: Distribución de un conductor con forma de toroide sin carga y con un campo externo de +10kV/m en dirección x.

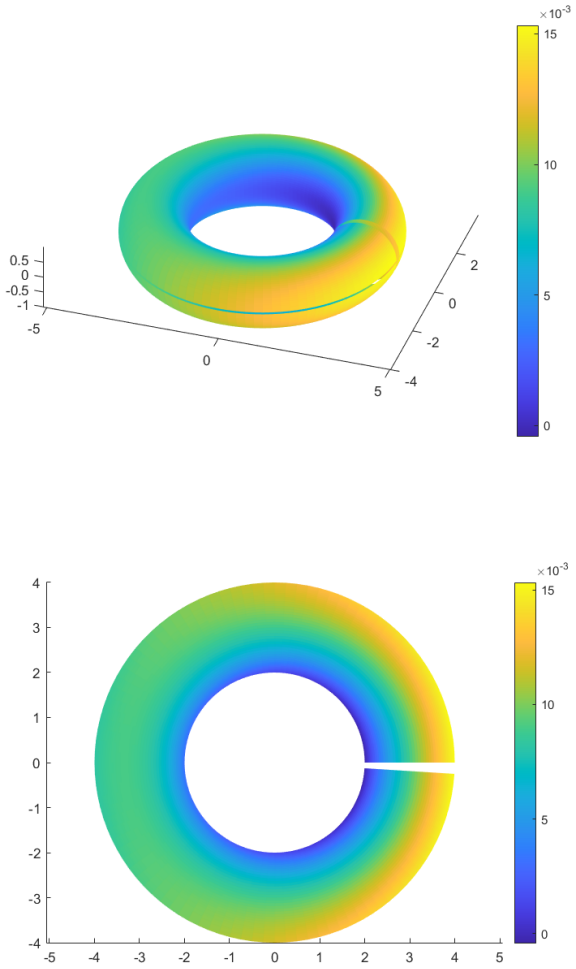


Figura 7: Distribución de un conductor con forma de toroide con una carga de 1C y campo externo de +10kV/m en dirección x.

4. Conclusiones

El método parece arrojar resultados coherentes para el caso particular del toroide y presenta algunos fallos para el caso de la esfera, por lo que se debe revisar el proceso y la metodología con el objetivo de hallar resultados más adecuados a la realidad.

Dado que una de las posibles fuentes de error es la forma de hacer la integral, una posible manera de mejorar el método es cambiar esto. Una opción es, una vez que se halla una nube de puntos evaluando la expresión que define la geometría, llevar a cabo un proceso de triangulado, donde cada triángulo se trataría como una carga puntual. Así se podría calcular geométricamente el área de cada triangulo y el vector normal a este. Además se podría considerar como posición del triángulo el de su baricentro (centro de carga).

A. Código programa

```
#include<iostream>
#include<stdio.h>
#include<math.h>

#define r 1.0
#define R 3.0

static const int Nu=60,Nv=60;
static const double pi=3.141592;
static double X[Nu][Nv], Y[Nu][Nv], Z[Nu][Nv];
static double VectorOrtogonal[Nu][Nv][3],Sactual[Nu][Nv],Santerior[Nu][Nv],NormadS[Nu][Nv];

///Devuelve el valor de un par metro u o v dado su indice
double conversionIndividual (int n, int N, double inicio, double fin)
{
    double total=fin-inicio;
    double Delta=total/N;
    return inicio+Delta*n;
}

///Devuelve el valor de (x,y,z) dado el valor de (u,v)
void cCoord(double u, double v, double &x, double &y, double &z, int geometria)
{
    switch(geometria)
    {
    case 1:
        x=r*cos(u)*sin(v);
        y=r*sin(u)*sin(v);
        z=r*cos(v);
        break;
    case 2:
        x=(R+r*cos(u))*cos(v);
        y=(R+r*cos(u))*sin(v);
        z=r*sin(u);
        break;
    }
}

///Devuelve (x,y,z) dado el  ndice  de u y v
void conversion (int nu, int nv,int geometria, double &x, double &y, double &z)
{

    switch(geometria)
    {
    case 1:
        cCoord(conversionIndividual(nu,Nu,0,2*pi),conversionIndividual(nv,Nv-1,0,pi),x,y,z,
    geometria);
        break;
    case 2:
        cCoord(conversionIndividual(nu,Nu,0,2*pi),conversionIndividual(nv,Nv,0,2*pi),x,y,z,
    geometria);
        break;

    }
}


double prodEscalar(double ux, double uy, double uz, double vx, double vy, double vz)
{
    return(ux*vx+uy*vy+uz*vz);
}

double norma(double x,double y, double z)
```

```
62 {
63     return(sqrt(prodEscalar(x,y,z,x,y,z)));
64 }
65
66 double coseno(double ux,double uy, double uz, double vx, double vy, double vz)
67 {
68     double cs=prodEscalar(ux,uy,uz,vx,vy,vz)/(norma(ux,uy,uz)*norma(vx,vy,vz));
69     return cs;
70 }
71
72 void prodVectorial(double ux,double uy, double uz, double vx, double vy, double vz, double & wx,
      double & wy, double & wz)
73 {
74     wx=uy*vz-uz*vy;
75     wy=-(ux*vz-uz*vx);
76     wz=ux*vy-uy*vx;
77 }
78
79 ///Devuelve (ux,uy,uz)-(vx,vy,vz)
80 void vectDiferencia(double ux,double uy, double uz, double vx, double vy, double vz, double & wx,
       double & wy, double & wz)
81 {
82     wx=ux-vx;
83     wy=uy-vy;
84     wz=uz-vz;
85 }
86
87 ///Devuelve el vector r (R-R') dados los indices de (u,v)
88 void vectDiferenciaIndices(int nu, int nv, int nu2, int nv2, double &vx,double &vy, double &vz)
89 {
90     vectDiferencia(X[nu][nv],Y[nu][nv],Z[nu][nv],X[nu2][nv2],Y[nu2][nv2],Z[nu2][nv2],vx,vy,vz);
91 }
92
93 void igualar()
94 {
95     int i,j;
96     for (i=0; i<Nu; i++)
97     {
98         for(j=0; j<Nv; j++)
99         {
100             Santerior[i][j]=Sactual[i][j];
101         }
102     }
103 }
104
105 ///Devuelve true si la desviaci n media respecto a la iteraci n anterior es menor que la
      precision fijada
106 bool compara(double precision)
107 {
108     double suma=0;
109     int i=0, j=0;
110     bool condicion;
111     for ( i=0; i<Nu; i++)
112     {
113         for( j=0; j<Nv; j++)
114         {
115             suma=suma + (Sactual[i][j]-Santerior[i][j])*(Sactual[i][j]-Santerior[i][j]);
116         }
117     }
118     suma=suma/((Nu)*(Nv));
119     if(suma<=precision)
120     {
121         condicion=true;
122     }
123     else
```

```
    {
        condicion=false;
    }
    return condicion;
}

///Crea los arrays X,Y,Z seg n la geometr a
void definirXYZ(int geometria)
{
    double x,y,z;
    int i=0,j=0;

    for(i=0; i<Nu; i++)
    {
        for(j=0; j<Nv; j++)
        {
            conversion(i,j,geometria,x,y,z);
            X[i][j]=x;
            Y[i][j]=y;
            Z[i][j]=z;
        }
    }


}

///Devuelve el valor de (u,v) dados sus indices
void cCoordDoble(int nu, int nv, double &u, double &v, int geometria)
{
    switch(geometria)
    {
    case 1:
        u=conversionIndividual(nu,Nu,0,2*pi);
        v=conversionIndividual(nv,Nv-1,0,pi);
        break;

    case 2:
        u=conversionIndividual(nu,Nu,0,2*pi);
        v=conversionIndividual(nv,Nv,0,2*pi);

    }
}

///Crea un array que contiene los vectores ortogonales para cada (x,y,z)
void crearVectorOrtogonal(int tipo)
{
    double xu,xv,yu,yv,zu,zv,u,v;
    double du,dv;
    switch (tipo)
    {
    case 1:
        du=2*pi/(Nu);
        dv=(pi)/(Nv-1);
        for(int i=0; i<Nu; i++)
        {
            for(int j=0; j<Nv; j++)
            {
                cCoordDoble(i,j,u,v,tipo);
                xu=-r*sin(v)*sin(u);
                xv=r*cos(u)*cos(v);
                yu=r*cos(u)*sin(v);
                yv=r*sin(u)*cos(v);
                zu=0;
                zv=-r*sin(v);
                prodVectorial(xv,yv,zv,xu,yu,zu,VectorOrtogonal[i][j][0],VectorOrtogonal[i][j
```

```
    ][1],VectorOrtogonal[i][j][2]);
                NormadS[i][j]=r*sin(v)*du*dv;
                ///NormadS[i][j]=norma(VectorOrtogonal[i][j][0],VectorOrtogonal[i][j][1],
    VectorOrtogonal[i][j][2])*du*dv;
                //printf("(%lf %lf) %lf %lf %lf, %lf %lf %lf\n",u,v, xu,yu,zu, xv, yv, zv);
            }
        }
        break;

    case 2:
        du=2*pi/(Nu);
        dv=2*pi/(Nv);
        for(int i=0; i<Nu; i++)
        {
            for(int j=0; j<Nv; j++)
            {
                cCoordDoble(i,j,u,v,tipo);
                xu=-cos(v)*r*sin(u);
                xv=-(R+r*cos(u))*sin(v);
                yu=-r*sin(v)*sin(u);
                yv=(R+r*cos(u))*cos(v);
                zu=r*cos(u);
                zv=0;
                prodVectorial(xv,yv,zv,xu,yu,zu,VectorOrtogonal[i][j][0],VectorOrtogonal[i][j
    ][1],VectorOrtogonal[i][j][2]);
                NormadS[i][j]=norma(VectorOrtogonal[i][j][0],VectorOrtogonal[i][j][1],
    VectorOrtogonal[i][j][2])*du*dv;

            }
        }

            }
        }
    }
}



void imprimirFichero(double matriz[][Nv],FILE*f)
{
    int i;
    for( i=0; i<Nu; i++)
    {
        for(int j=0; j<Nv; j++)
        {
            fprintf(f,"%lf\t",matriz[i][j]);
        }
        // fprintf(f,"%lf\t",matriz[i][0]);
        fprintf(f,"\n");
    }
    i=0;
    /**for(int j=0;j<Nv;j++){
                fprintf(f,"%lf\t",matriz[i][j]);
        }
             fprintf(f,"%lf\t",matriz[i][0]);
        fprintf(f,"\n");**/
}

void integral(int nu, int nv,double Ex, double Ey, double Ez, int geometria)
{
    int i,j;
    double aux,rx,ry,rz;
    Sactual[nu][nv]=0.0;
    //printf("%lf",Sactual[nu][nv]);
    switch(geometria)
```

```
    {
    case 1:
        if((nv==0||nv==(Nv-1))&&nu!=0)
        {
            Sactual[nu][nv]=Sactual[0][nv];

        }
        else if((nv==0||nv==(Nv-1))&&nu==0)
        {
            for(i=0; i<Nu; i++)
            {
                for(j=0; j<Nv; j++)
                {
                    if((i!=nu||j!=nv)&&(j!=0&&j!=(Nv-1)))
                    {
                        vectDiferenciaIndices(nu,nv,i,j,rx,ry,rz);
                        aux=Sactual[nu][nv];

                        Sactual[nu][nv]=aux+1/(36*pi)*pow(10,-9)*prodEscalar(Ex,Ey,Ez,
    VectorOrtogonal[nu][nv][0],VectorOrtogonal[nu][nv][1],VectorOrtogonal[nu][nv][2])+1/(2*pi*(
    norma(rx,ry,rz)*norma(rx,ry,rz)))*coseno(X[nu][nv],Y[nu][nv],Z[nu][nv],rx,ry,rz)*NormadS[i][j
    ]*Santerior[i][j];

                    }
                    else if(((nv==0&&j==(Nv-1))||(nv==(Nv-1)&&j==0))&&i==0)
                    {
                        vectDiferenciaIndices(nu,nv,i,j,rx,ry,rz);
                        aux=Sactual[nu][nv];

                        Sactual[nu][nv]=aux+1/(36*pi)*pow(10,-9)*prodEscalar(Ex,Ey,Ez,
    VectorOrtogonal[nu][nv][0],VectorOrtogonal[nu][nv][1],VectorOrtogonal[nu][nv][2])+1/(2*pi*(
    norma(rx,ry,rz)*norma(rx,ry,rz)))*coseno(X[nu][nv],Y[nu][nv],Z[nu][nv],rx,ry,rz)*NormadS[i][j
    ]*Santerior[i][j];

                    }

                }
            }
        }
        else
        {
            for(i=0; i<Nu; i++)
            {
                for(j=0; j<Nv; j++)
                {
                    if((i!=nu||j!=nv)&&(j!=0&&j!=(Nv-1)))
                    {
                        vectDiferenciaIndices(nu,nv,i,j,rx,ry,rz);
                        aux=Sactual[nu][nv];

                        Sactual[nu][nv]=aux+1/(36*pi)*pow(10,-9)*prodEscalar(Ex,Ey,Ez,
    VectorOrtogonal[nu][nv][0],VectorOrtogonal[nu][nv][1],VectorOrtogonal[nu][nv][2])+1/(2*pi*(
    norma(rx,ry,rz)*norma(rx,ry,rz)))*coseno(X[nu][nv],Y[nu][nv],Z[nu][nv],rx,ry,rz)*NormadS[i][j
    ]*Santerior[i][j];

                    }
                    else if(i==0&&(j==0||j==(Nv-1)))
                    {
                        vectDiferenciaIndices(nu,nv,i,j,rx,ry,rz);
                        aux=Sactual[nu][nv];

                        Sactual[nu][nv]=aux+1/(36*pi)*pow(10,-9)*prodEscalar(Ex,Ey,Ez,
    VectorOrtogonal[nu][nv][0],VectorOrtogonal[nu][nv][1],VectorOrtogonal[nu][nv][2])+1/(2*pi*(
    norma(rx,ry,rz)*norma(rx,ry,rz)))*coseno(X[nu][nv],Y[nu][nv],Z[nu][nv],rx,ry,rz)*NormadS[i][j
    ]*Santerior[i][j];
```

```
303
304                     }
305
306                 }
307             }
308         }
309         break;
310
311     case 2:
312
313         for(i=0; i<Nu; i++)
314         {
315             for(j=0; j<Nv; j++)
316             {
317                 if(i!=nu||j!=nv)
318                 {
319                     vectDiferenciaIndices(nu,nv,i,j,rx,ry,rz);
320                     aux=Sactual[nu][nv];
321
322                     Sactual[nu][nv]=aux+1/(36*pi)*pow(10,-9)*
323                     prodEscalar(Ex,Ey,Ez,VectorOrtogonal[nu][nv][0],
324                     VectorOrtogonal[nu][nv][1],VectorOrtogonal[nu][nv][2])+1/(2*pi*(norma(rx,ry,
    rz)*norma(rx,ry,rz)))*
325                     coseno(VectorOrtogonal[nu][nv][0],VectorOrtogonal[nu][nv][1],VectorOrtogonal[
    nu][nv][2],rx,ry,rz)*NormadS[i][j]*Santerior[i][j];
326                      %lf\n",i,j,Sactual[nu][nv],aux,norma(rx,ry,rz),VectorOrtogonal[nu][nv][0],
327                      VectorOrtogonal[nu][nv][1],VectorOrtogonal[nu][nv][2],
328                      NormadS[i][j],coseno(rx,ry,rz,VectorOrtogonal[nu][nv][0],VectorOrtogonal[nu
    ][nv][1],VectorOrtogonal[nu][nv][2]),1/(2*pi*(norma(rx,ry,rz)*
329                      norma(rx,ry,rz)))*coseno(rx,ry,rz,VectorOrtogonal[nu][nv][0],VectorOrtogonal
    [nu][nv][1],
330                      VectorOrtogonal[nu][nv][2])*NormadS[i][j]);
331                 }
332             }
333         }
334     }
335     //printf("Sactual: %lf\n",Sactual[nu][nv]);
336
337 }
338
339
340
341 int main()
342 {
343
344     int geometria=2;
345
346     double Ex=1000000,Ey=0,Ez=0;
347
348     FILE *fx=fopen("coordenadasX.txt","w"),*fy=fopen("coordenadasY.txt","w"),*fz=fopen("
    coordenadasZ.txt","w"),*fs=fopen("valoresS.txt","w"),*fDx=fopen("vectDx.txt","w"),*fDy=fopen(
    "vectDy.txt","w"),*fDz=fopen("vectDz.txt","w");
349
350     double precision=0.001;
351     definirXYZ(geometria);
352     crearVectorOrtogonal(geometria);
353     imprimirFichero(X,fx);
354     imprimirFichero(Y,fy);
355     imprimirFichero(Z,fz);
356     int i,j;
357     for(i=0; i<Nu; i++)
358     {
359         for(j=0; j<Nv; j++)
360         {
361             fprintf(fDx,"%lf\t",VectorOrtogonal[i][j][0]);
```

```
        }
        fprintf(fDx,"\n");
    }
    for(i=0; i<Nu; i++)
    {
        for(j=0; j<Nv; j++)
        {
            fprintf(fDy,"%lf\t",VectorOrtogonal[i][j][1]);
        }
        fprintf(fDy,"\n");
    }
    for(i=0; i<Nu; i++)
    {
        for(j=0; j<Nv; j++)
        {
            fprintf(fDz,"%lf\t",VectorOrtogonal[i][j][2]);
        }
        fprintf(fDz,"\n");
    }
    int f=0;
    double Qtotal=10.0;
    printf("for");

    double densidadInicial=Qtotal/(4*pi*r*r);

    for(i=0; i<Nu; i++)
    {
        for(j=0; j<Nv; j++)
        {
            Santerior[i][j]=0;
            Sactual[i][j]=densidadInicial;
        }
    }
    while(!compara(precision))
    {
        printf("vuelta %d\n",f);
        igualar();
        for(i=0; i<Nu; i++)
        {
            for(j=0; j<Nv; j++)
            {
                integral(i,j,Ex,Ey,Ez,geometria);
            }
        }
        f++;
    }

    imprimirFichero(Sactual,fs);


    fclose(fx);
    fclose(fy);
    fclose(fz);
    fclose(fs);
}
```

Listing 1: Código

B. Código representación gráfica en MATLAB

```

X=importdata("coordenadasX.txt");
Y=importdata("coordenadasY.txt");
```

```
Z=importdata("coordenadasZ.txt");

figure
S=importdata("valoresS.txt");
hold on
surf(X,Y,Z,S)
axis equal
hold off
colorbar
shading flat
```

Listing 2: Código para prepresentación gráfica en matlab.