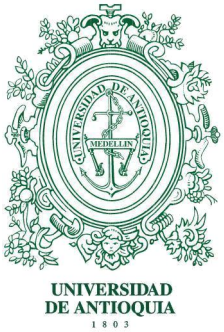
Suplemento Computacional

Electricidad y Magnetismo

Sebastian Bustamante Jaramillo

macsebas33@gmail.com



Facultad de Ciencias Exactas y Naturales
Universidad de Antioquia

Índice general

Capítulo 1

Preliminares

1.1. Motivación

La física ha evolucionado hasta un estado actual donde la mayoría de cálculos teóricos necesarios para realizar investigación de frontera requieren de una gran componente computacional. Desde la corroboración entre teoría y experimento, la predicción y control de los resultados de un experimento hecho a posteriori y la recreación de condiciones imposibles de lograr experimentalmente, tales como simulaciones cosmológicas del universo a gran escala o complejos sistemas atómicos. Estos son sólo algunos ejemplos representativos del papel de la computación en la física moderna. Debido a esto, el principal objetivo del suplemento computacional es la introducción temprana en los cursos de física básica de herramientas computacionales que serán de utilidad a los estudiantes en este curso específico y durante el transcurso de sus carreras científicas.

1.2. Instalación de Paquetes

En la totalidad de esta guía será usado el lenguaje de programación *Python* como referente para todos las prácticas y ejercicios computaciones. La principal motivación de esto es su facilidad de implementación en comparación a otros lenguajes también de amplio en ciencia. Además es un lenguaje interpretado, lo que permite una depuración más sencilla por parte del estudiante, sin necesidad de usar más complicados sistemas de depuración en el caso de lenguajes compilados como C o Fortran. *Python* es un lenguaje de código abierto, lo que permite la libre distribución del paquete y evita el pago de costosas licencias de uso, además la gran mayoría de paquetes que extienden enormemente la funcionalidad de *Python* son también código abierto y de libre distribución y uso.

A pesar de que *Python* es un lenguaje multiplataforma, permitiendo correr scripts python en Linux, Windows y Mac, acá solo se indicará el método de instalación para distribuciones Linux basadas en Debian.

La última versión de *Python* de la rama 2 es 2.7.4 y de la rama 3 es la 3.3.1, debido a ligeras incompatibilidades entre ambas ramas de desarrollo, será utilizada la rama 2 en una de sus últimas versiones. En orden, para instalar *Python* en una versión Linux basta con descargarlo directamente de los repositorios oficiales[1], en el caso de una distro basada en Debian el gestor de paquetes es `apt-get`, y desde una terminal se tiene

```
\$ apt-get install python2.7
```

también puede descargarse directamente desde la página oficial del proyecto `http://python.org/`.

Una vez instalada la última versión de *Python* , es necesario instalar los siguiente paquetes para el correcto desarrollo de las aplicaciones del curso:

iPython

iPython es un shell que permite una interacción más interactiva con los scripts de python, permitiendo el resaltado de sintaxis desde consola, funciones de autocompletado y depuración de código más simple. Para su instalación basta descargarlo de los repositorios oficiales

```
\$ apt-get install ipython
```

o puede de descargarse de la página oficial `http://ipython.org/`. También puede encontrarse documentación completa y actualizada en esta página, se recomienda visitarla frecuentemente para tener las más recientes actualizaciones.

NumPy

NumPy es una librería que extiende las funciones matemáticas de *Python* , permitiendo el manejo de matrices y vectores. Es esencial para la programación científica en *Python* y puede ser instalada de los repositorios

```
\$ apt-get install python-numpy
```

[1] En la mayoría de distribuciones Linux *Python* viene precargado por defecto.

La última versión estable es la 1.6.2. En la página oficial del proyecto puede encontrarse versiones actualizadas y una amplia documentación `http://www.numpy.org/`.

SciPy

SciPy es una amplia biblioteca de algoritmos matemáticos para *Python* , esta incluye herramientas que van desde funciones especiales, integración, optimización, procesamiento de señales, análisis de Fourier, etc. Al igual que los anteriores paquetes, puede ser instalada desde los repositorios oficiales

```
\$ apt-get install python-scipy
```

Una completa documentación del paquete puede ser encontrada en `http://docs.scipy.org/doc/scipy/reference/`. La última versión estable es la 0.11.0 y puede ser encontrada en la página oficial del proyecto `http://www.scipy.org/`.

Matplotlib

Matplotlib es una completa librería con rutinas para la generación de gráficos a partir de datos. Aunque en su estado actual está enfocada principalmente a gráficos 2D, permite un amplio control sobre el formato de las gráficas generadas, dando una amplia versatilidad a los usuarios. Su instalación puede realizarse a partir de los repositorios oficiales

```
\$ apt-get install python-matplotlib
```

La última versión estable es la 1.2.1. y puede encontrarse en la página oficial del proyecto `http://matplotlib.org/`. Una amplia documentación está disponible en `http://matplotlib.org/1.2.0/contents.html`.

MayaVi2

MayaVi2 es una librería para la visualización científica en python, en especial para gráficos 3D, permitiendo funciones avanzadas como renderizado, manejo de texturas, etc. Se encuentra en los repositorios oficiales

```
\$ apt-get install mayavi2
```

La versión 2 es una versión mejorada de la original, estando más orientada a la reutilización de código. Por defecto incluye una interfaz gráfica que facilita su manejo. La página oficial del proyecto es `http://mayavi.sourceforge.net/`.

Tkinter

TKinter es una librería para la gestión gráfica de aplicaciones in *Python* y viene por defecto instalada, aún así puede ser instalada de los repositorios oficiales

```
\$ apt-get install python-tk
```

La página oficial del proyecto es `http://wiki.python.org/moin/TkInter`. Para el desarrollo de entornos gráficos existen otras llamativas alternativas como PyGTK o PyQt, pero debido a la facilidad de uso y a ser la librería estándar soportada, *TKinter* será usada en este curso.

1.3. Ejemplo de Uso

En esta sección se ilustra un ejemplo sencillo que permite al estudiante identificar la manera estándar de ejecutar códigos en *Python* , además probar los paquetes instalados.

El código de ejemplo permite graficar dos funciones diferentes en una misma ventana y además un conjunto de datos aleatorios generados en el eje Y.

```
1  #!/usr/bin/env python
2  #============================================================
3  # EJEMPLO DE USO
4  # Grafica de funciones y datos aleatorios
5  #============================================================
6  import numpy as np
7  import scipy as sp
8  import matplotlib.pylab as plt
9
10 #Funcion 1
11 def Funcion1(x):
12     f1 = np.sin(x)/( np.sqrt(1 + x**2) )
13     return f1
14
15 #Funcion 2
16 def Funcion2(x):
```

```
17      f2 = 1/(1+x)
18      return f2
19
20  #Valores de x para evaluar
21  X = np.linspace( 0, 10, 100 )
22  #Evaluacion de funcion 1
23  F1 = Funcion1(X)
24  #Evaluacion de funcion 2
25  F2 = Funcion2(X)
26
27  #Grafica funcion 1
28  plt.plot( X, F1, label='Funcion 1' )
29  #Grafica funcion 2
30  plt.plot( X, F2, label='Funcion 2' )
31
32  #Datos aleatorios eje Y
33  Yrand = sp.random.rand( 100 )
34  #Grafica datos aleatorios
35  plt.plot( X, Yrand, 'o', label='Datos' )
36
37  plt.legend()
38  plt.show()
```

El resultado obtenido es la siguiente gráfica

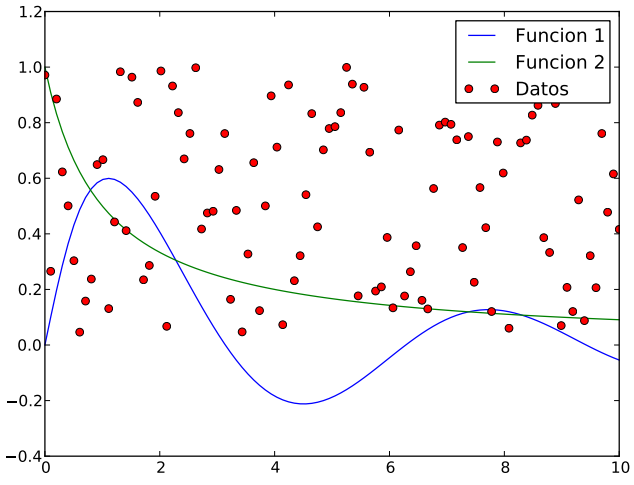


Figura 1.1: Resultado del ejemplo anterior, gráfica de dos funciones y datos aleatorios.

Para obtener el anterior script, el estudiante puede transcribirlo directamente de esta guía o puede descargarlo del repositorio oficial del curso[2] en el link `https://github.com/sbustamante/Computacional-Campos/raw/master/codigos/usage_01.py`. Una vez obtenido el archivo `usage_01.py`, abrir una terminal en la carpeta donde se ha guardado y escribir `ipython` para abrir el intérprete de *Python*

```
\$ ipython
```

Se debe obtener algo como

```
\$ ipython
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]:
```

Finalmente para ejecutar el script, usar el comando `run` seguido del nombre del código en el intérprete de código *Python*

```
In [1]: run usage_01.py
```

[2] Repositorio oficial en `https://github.com/sbustamante/Computacional-Campos`

Capítulo 2

Electrostática

La electrostática estudia la interacción entre cuerpos cargados eléctricamente sin tener en cuenta su movimiento. Esta simplificación permite ignorar términos dinámicos asociados a corrientes eléctricas y campos magnéticos inducidos.

En este capítulo se realizan algunas demostraciones computacionales que van desde el cálculo de trayectorias de partículas en campos eléctricos y magnéticos, la representación de las líneas de campo y superficies equipotenciales de distribuciones de carga, hasta el cálculo de capacitancia de algunos sistemas.

2.1. Demostración 1: Espectrómetro de Masas

En esta primera demostración será estudiado el espectrómetro de masas. El objetivo de este dispositivo es caracterizar partículas cargadas de acuerdo a su relación carga masa (q/m). Su funcionamiento consiste en la inmersión de las partículas en un campo magnético o eléctrico (este caso) y a partir de las trayectorias obtenidas determinar su relación carga masa (ver figura 2.1).

Tomando una partícula de masa m y carga q embebida en un campo eléctrico homogéneo y uniforme $\boldsymbol{E}$, la ecuación de movimiento es

$$m\frac{d^2\boldsymbol{r}}{dt^2} = q\boldsymbol{E} \tag{2.1}$$

Tomando el sistema coordenado de tal forma que el campo $\boldsymbol{E}$ esté en la dirección positiva de y, se obtiene

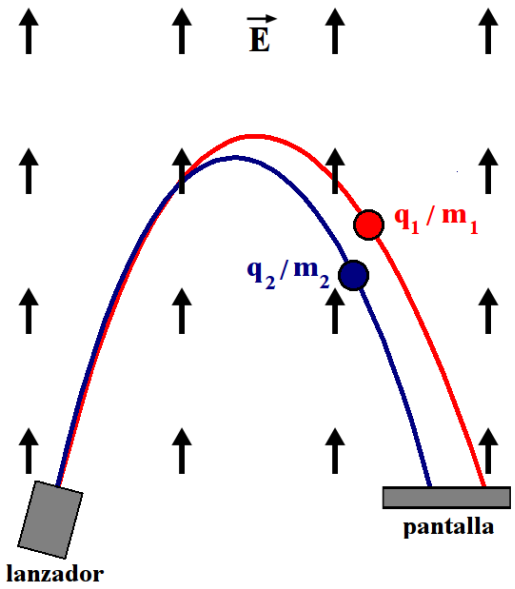


Figura 2.1: Espectrómetro de masas.

$$x(t) = x_0 + v_{x0}t \tag{2.2}$$

$$y(t) = y_0 + v_{y0}t + \frac{1}{2}\left(\frac{q}{m}\right)t^2 \tag{2.3}$$

donde se ha introducido la posición inicial de la partícula $\boldsymbol{r}(t=0) = (x_0, y_0)$ y la velocidad inicial $\boldsymbol{v}(t=0) = (v_{x0}, v_{y0})$.

Eliminando el tiempo de las dos ecuaciones se obtiene la siguiente trayectoria

$$y(x) = y_0 + \frac{v_{y0}}{v_{x0}}(x - x_0) + \frac{1}{2}\left(\frac{q}{m}\right)\left(\frac{x - x_0}{v_{x0}}\right)^2 \tag{2.4}$$

En el siguiente código de *Python* se grafica la trayectoria de dos partículas con diferente relación carga masa. La primera tiene una relación $q_1/m_1 = -1\,\mathrm{C}/1\,\mathrm{kg}$ y la segunda $q_2/m_2 = -1\,\mathrm{C}/2\,\mathrm{kg}$. Ambas partículas se disparan del origen y con una elocidad inicial de $\boldsymbol{v}_0 = (1, 2)$ m/s. El campo eléctrico tiene una intensidad de $|\boldsymbol{E}| = 1$ N/C.

```
#!/usr/bin/env python
#============================================================
# DEMOSTRACION 1
# Espectrometro de masas
#============================================================
from __future__ import division
import numpy as np
import matplotlib.pylab as plt

#Trayectoria
def trayectory(x):
    y = y0 + vy0/vx0*(x - x0) + 0.5*(q/m)*( (x-x0)/vx0 )**2
    return y

#PARTICULA 1
#Carga
q = -1
#Masa
m = 1
#Posicion inicial
x0 = 0
y0 = 0
#Velocidad inicial
vx0 = 1
vy0 = 2
#Valores de X a graficar
X = np.arange( 0, 10, 0.01 )
#Trayectoria
Y = trayectory( X )
#Grafica de trayectoria
plt.plot( X, Y, label='particula 1' )

#PARTICULA 2
#Carga
q = -1
#Masa
m = 2
#Posicion inicial
x0 = 0
y0 = 0
#Velocidad inicial
vx0 = 1
```

```
43 vy0 = 2
44 #Valores de X a graficar
45 X = np.arange( 0, 10, 0.01 )
46 #Trayectoria
47 Y = trayectory( X )
48 #Grafica de trayectoria
49 plt.plot( X, Y, label='particula 2' )
50
51 #Limites del eje X
52 plt.xlim( (0,10) )
53 #Limites del eje Y
54 plt.ylim( (0,10) )
55 plt.legend()
56 plt.show()
```

El resultado que se obtiene es

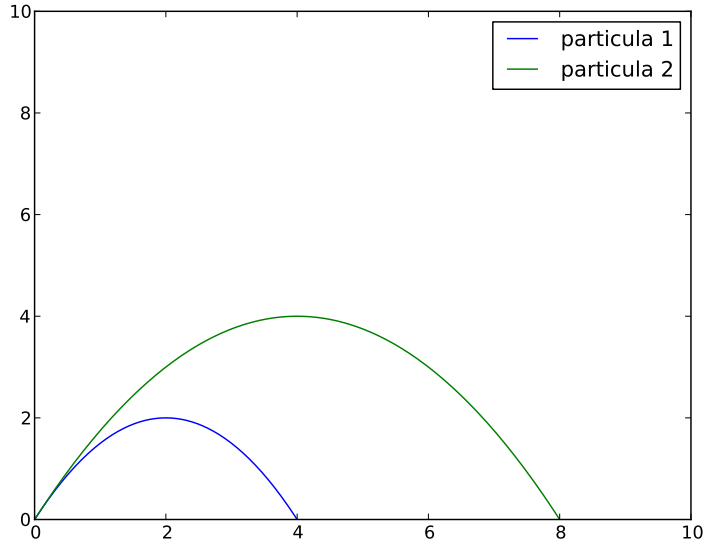


Figura 2.2: Trayectorias según la relación carga masa de las partículas.

La representación de la trayectoria de forma gráfica permite entonces comparar directamente con datos medidos en un laboratorio, por ejemplo trayectorias de partículas en cámaras de burbujas.

A continuación se explica cada componente del código anterior

```
from __future__ import division
import numpy as np
import matplotlib.pylab as plt
```

En la primera línea se carga el módulo `division`, este permite a *Python* calcular fracciones de números enteros como cantidades reales. En la siguiente línea se carga la librería *NumPy* con el alias de `np` y finalmente se carga la librería *Matplotlib* con el alias de `plt`.

```
#Trayectoria
def trayectory(x):
    y = y0 + vy0/vx0*(x - x0) + 0.5*(q/m)*( (x-x0)/vx0 )**2
    return y
```

En esta parte se define la trayectoria de la partícula en el campo eléctrico descrito.

```
#PARTICULA 1
#Carga
q = -1
#Masa
m = 1
#Posicion inicial
x0 = 0
y0 = 0
#Velocidad inicial
vx0 = 1
vy0 = 2
#Valores de X a graficar
X = np.arange( 0, 10, 0.01 )
#Trayectoria
Y = trayectory( X )
#Grafica de trayectoria
plt.plot( X, Y, label='particula 1' )
```

Se definen las propiedades físicas y cinemáticas de la partícula 1. Su carga, su masa, su posición y velocidad inicial. Luego, usando el comando `arange` de la librería *NumPy* , se construye un arreglo de valores en el eje X que serán usados para el cálculo de la trayectoria. En este caso se toma desde 0 a 10 m con un salto de 0,01 m. Finalmente se llama la función de la trayectoria de la partícula en todos los valores de `X` y se grafica, usando como etiqueta `label='particula 1'`.

```
#Limites del eje X
plt.xlim( (0,10) )
#Limites del eje Y
plt.ylim( (0,10) )
plt.legend()
plt.show()
```

Finalmente se usan las funciones de *Matplotlib* `xlim` y `ylim` para fijar los límites de la ventana de graficación. La función `legend` muestra las etiquetas de las dos trayectorias y finalmente `show` muestra en pantalla el resultado.

2.2. Demostración 2: Líneas de Campo y Equipotenciales

Un campo es una distribución espacial de alguna cantidad física, tal como el campo eléctrico y magnético. Este puede ser generado por una fuente, tal como es el caso del campo electrostático generado por una distribución de cargas, o pueden no tener una fuente asociada, como el caso de ondas electromagnéticas que viajan libre en el espacio.
En esta demostración se calculará las líneas de campo eléctrico y las líneas de equipotencial para una distribución de dos partículas puntuales. Debido al principio de superposición se tiene entonces

$$\begin{aligned} \varphi_{tot}(\boldsymbol{r}) &= \varphi_1(\boldsymbol{r}) + \varphi_2(\boldsymbol{r}) \\ &= \frac{1}{4\pi\epsilon_0}\frac{q_1}{|\boldsymbol{r}-\boldsymbol{r}_1|} + \frac{1}{4\pi\epsilon_0}\frac{q_2}{|\boldsymbol{r}-\boldsymbol{r}_2|} \end{aligned} \tag{2.5}$$

$$\begin{aligned} \boldsymbol{E}_{tot}(\boldsymbol{r}) &= \boldsymbol{E}_1(\boldsymbol{r}) + \boldsymbol{E}_2(\boldsymbol{r}) \\ &= \frac{1}{4\pi\epsilon_0}\frac{q_1}{|\boldsymbol{r}-\boldsymbol{r}_1|^3}(\boldsymbol{r}-\boldsymbol{r}_1) + \frac{1}{4\pi\epsilon_0}\frac{q_2}{|\boldsymbol{r}-\boldsymbol{r}_2|^3}(\boldsymbol{r}-\boldsymbol{r}_2) \end{aligned} \tag{2.6}$$

El script en *Python* para realizar la demostración es:

```
1 #!/usr/bin/env python
2 #============================================================
3 # DEMOSTRACION 2
```

```
4  # Lineas de campo y equipotenciales de cargas puntuales
5  #============================================================
6  from __future__ import division
7  import numpy as np
8  import matplotlib.pylab as plt
9
10 #Potencial de una particula puntual cargada
11 def Phi( r, rp, q ):
12     phi = 1/(4*np.pi*eps0)*q/np.linalg.norm( r - rp )
13     return phi
14
15 #Campo electrico de una particula puntual cargada
16 def Electric( r, rp, q ):
17     E = 1/(4*np.pi*eps0)*q*(r - rp)/np.linalg.norm( r - rp )
           **3
18     return E
19
20 #Permitividad del vacio
21 eps0 = 8.85418e-12
22 #Resolucion de graficas
23 Nres = 20
24 #Coordenada X
25 Xarray = np.linspace( 0, 10, Nres )
26 #Coordenada Y
27 Yarray = np.linspace( 0, 10, Nres )
28 #Construccion de la cuadricula
29 X, Y = plt.meshgrid( Xarray, Yarray )
30
31 #CONSTRUCCION DE CAMPO E Y POTENCIAL PHI, PARTICULA 1
32 #Carga electrica
33 q1 = -1
34 #Posicion particula
35 rp1 = np.array( [4,4] )
36 #Incializacion Potencial Electrico
37 phi1 = np.zeros( (Nres,Nres) )
38 #Incializacion Campo Electrico
39 E1x = np.ones( (Nres,Nres) )
40 E1y = np.ones( (Nres,Nres) )
41 #Calculo Potencial Electrico y Campo Electrico
42 for i in xrange(Nres):
43     for j in xrange(Nres):
44         r = np.array( [Xarray[i], Yarray[j]] )
45         phi1[i,j] = Phi( r, rp1, q1 )
```

```
46          E = Electric( r, rp1, q1 )
47          E1x[i,j], E1y[i,j] = E/np.linalg.norm(E)
48
49  #CONSTRUCCION DE CAMPO E Y POTENCIAL PHI, PARTICULA 2
50  #Carga electrica
51  q2 = -1
52  #Posicion particula
53  rp2 = np.array( [6,6] )
54  #Incializacion Potencial Electrico
55  phi2 = np.zeros( (Nres,Nres) )
56  #Incializacion Campo Electrico
57  E2x = np.ones( (Nres,Nres) )
58  E2y = np.ones( (Nres,Nres) )
59  #Calculo Potencial Electrico y Campo Electrico
60  for i in xrange(Nres):
61      for j in xrange(Nres):
62          r = np.array( [Xarray[i], Yarray[j]] )
63          phi2[i,j] = Phi( r, rp2, q2 )
64          E = Electric( r, rp2, q2 )
65          E2x[i,j], E2y[i,j] = E/np.linalg.norm(E)
66
67  #CONSTRUCCION DE CAMPO E Y POTENCIAL PHI, TOTAL
68  phi_tot = phi1 + phi2
69  #Incializacion Campo Electrico
70  Ex_tot = np.ones( (Nres,Nres) )
71  Ey_tot = np.ones( (Nres,Nres) )
72  #Calculo Potencial Electrico y Campo Electrico Total
73  for i in xrange(Nres):
74      for j in xrange(Nres):
75          E = np.array( [E1x[i,j] + E2x[i,j], \
76          E1y[i,j] + E2y[i,j]] )
77          E = E/np.linalg.norm(E)
78          Ex_tot[i,j], Ey_tot[i,j] = E
79
80  #Grafica de equipotenciales
81  plt.contour(X, Y, phi_tot, 50)
82  #Grafica de lineas de campo
83  plt.quiver( X, Y, Ey_tot, Ex_tot)
84
85  #Limites del eje X
86  plt.xlim( (0,10) )
87  #Limites del eje Y
88  plt.ylim( (0,10) )
```

```
89 plt.legend()
90 plt.show()
```

El resultado obtenido es

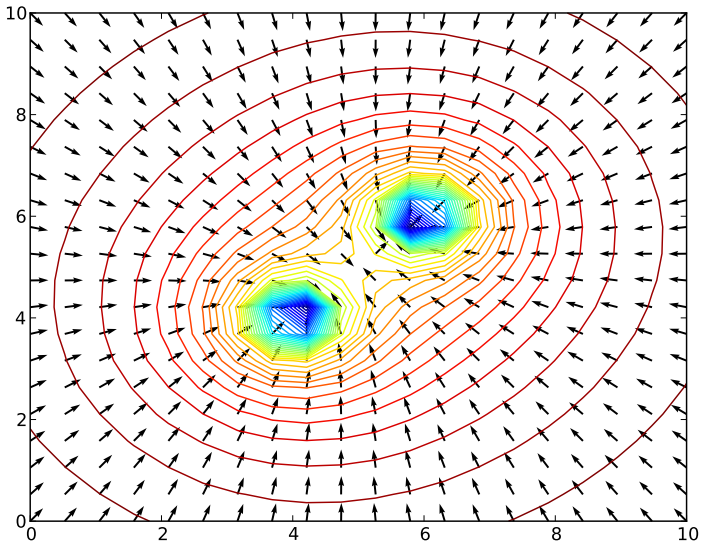


Figura 2.3: Líneas de campo y equipotenciales de dos cargas puntuales.

Capítulo 3

Bibliografía