

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

INFORMACIÓN BÁSICA					
ASIGNATURA:	<b>Física Computacional.</b>				
TÍTULO DE LA PRÁCTICA:	<b>Práctica de Ecuación diferencial de Laplace.</b>				
NÚMERO DE PRÁCTICA:	<b>05</b>	AÑO LECTIVO:	<b>2022-A</b>	NRO. SEMESTRE:	<b>VII</b>
FECHA DE PRESENTACIÓN:	June 11, 2022	HORA DE PRESENTACIÓN:	<b>10:45 pm</b>		
Integrante(s): <b>Alván Ventura Edsel Yael</b>				NOTA	
DOCENTE(s): <b>Danny Giancarlo Apaza Veliz.</b>					

## Práctica 5

### Física Computacional

Escrito por  
Alván Ventura, Edsel Yael  
ealvan@unsa.edu.pe

Profesor  
Apaza Veliz, Danny Giancarlo  
dapazav@unsa.edu.pe

June 11, 2022

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

## 1 Problema 1

Resuelva aproximadamente la ecuación de Laplace  $\nabla^2 u = 0$  en un cuadrado definido por  $0 < x < 4$  y  $0 < y < 4$ , con las condiciones en la frontera.

$$\begin{aligned} u(x, 0) &= 20 \text{ y } u(x, 4) = 300 \text{ para todo } 0 < x < 4 \\ u(0, y) &= 80 \text{ y } u(4, y) = 0 \text{ para todo } 0 < y < 4 \end{aligned}$$

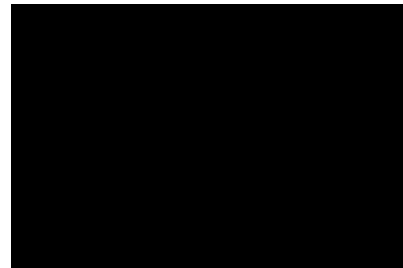
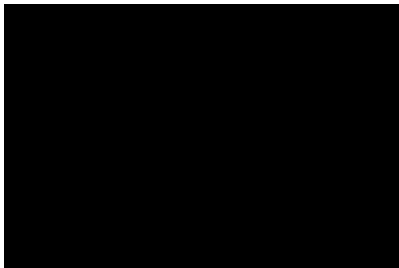
### 1.1 Análisis

La ecuación diferencial de Laplace puede ser resuelta numericamente mediante la técnica de diferencias finitas, las diferencias finitas pueden ayudarnos a aproximar mucho a una solución analítica.

Para lograrlo, necesitamos deducir esta formula:

$$u(i, j) = \frac{1}{4}(u_{i+1,j} + u_{i-1,j} + u_{i,j-1} + u_{i,j+1}) \quad (1)$$

A partir de las series de Taylor, para deducir una aproximación numerica a la segunda derivada de la funcion  $u(i, j)$ :



### 1.2 Programación

Luego de la deducción:

1. Se creó una matriz de  $m \times n$ .
2. Los contornos de la matriz se llenaron con los bordes dados por la función.
3. Luego la parte central de la matriz(no incluyendo los bordes) se lleno de la media de los valores de los valores del borde.
4. Despues de hacer eso, se implementó la técnica numérica para iterar sobre la parte central de la matriz anteriormente dicha.
5. Y por ultimo, se coloco la matriz en un mapa de calor, para ver las condiciones frontera.

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

Las dimensiones del array son parametros de la función llamada *Laplace()*. Esta función se denota de la siguiente manera:

$$Laplace(xu, xd, yl, yr, f, c, h) \quad (2)$$

Donde:

Parámetros:	Descripción
xu:	Representa la recta $u(x, 4) = 300 = xu$ arriba del cuadrado
xd:	Representa la recta $u(x, 0) = 20 = xd$ abajo del cuadrado
yl:	Representa la recta $u(0, y) = 80 = yl$ izquierda del cuadrado
yr:	Representa la recta $u(4, y) = 300 = yr$ derecha del cuadrado
f:	Representa el numero de filas de la matriz
c:	Representa el numero de columnas de la matriz
h:	Representa el numero de iteraciones de la matriz

A continuación se muestra la implementación:

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 #LAPLACE ->
4 #Ua= Yl
5 #Ub= Yr
6 #Uc= Xd
7 #Ud= Xu
8 #N = a
9 #M = b
10 #H = 1000 iterations
11 #Error: 0.0001 error
12 # def mean() hara el trabajo de media
13
14 def fillMatrix(arr, xu, xd, yl, yr):
15     arr[0, :] = xu
16     arr[arr.shape[0] - 1, :] = xd
17     arr[:, 0] = yl
18     arr[:, arr.shape[1] - 1] = yr
19     arr[1, :arr.shape[1] - 2]
20     arr[1:arr.shape[0] - 1, 1:arr.shape[1] - 1] = (xu+xd+yl+yr)/4
21
22 def calculate(arr, times, error):
23     convergencia = 0

```

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

```

24     contador = 0
25     # tmp = arr
26     while(contador < times and convergencia == 0):
27         tmp = arr
28         for i in range(1, arr.shape[0] - 1):
29             for j in range(1, arr.shape[1] - 1):
30                 arr[i, j] = 0.25*( arr[i + 1][j] + arr[i - 1][j] + arr[i][j + 1] + arr[i][j - 1])
31                 # if(np.linalg.norm(arr - tmp, np.inf) / np.linalg.norm(arr, np.inf) < error):
32                 #     convergencia = 1
33                 contador += 1
34     # if(convergencia == 1):
35     getCalorMap(arr)
36     print(f"contador: {contador} \n [arr - tmp] = {np.linalg.norm(tmp)} \n [arr] = {np.linalg.norm(arr)}")
37     # print(arr[:int(arr.shape[0]/2), :int(arr.shape[1]/2)])
38
39 def getCalorMap(arr):
40     plt.imshow(arr, cmap='hot', interpolation='nearest')
41     plt.show()
42
43 def Laplace(arriba, abajo, izq, dere, filas, cols, h, error):
44     malla = np.zeros((filas, cols)) # creando malla
45     fillMatrix(malla, arriba, abajo, izq, dere) # llenando malla
46     calculate(malla, h, error)
47
48 def main():
49     filas = 100
50     cols = 100
51     x_upper = 0
52     x_down = 80
53     l_left = 20
54     l_right = 300
55     times = 500
56     error = 0.001
57     Laplace(x_upper, x_down, l_left, l_right, filas, cols, times, error)
58     # print(malla)
59
60 if __name__ == "__main__":
61     main()

```

Como se puede ver, los datos del primer ejercicio estan puestos en la siguiente pieza de código:

```

1 def main():
2     filas = 100
3     cols = 100
4     x_upper = 0
5     x_down = 80
6     l_left = 20
7     l_right = 300

```

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

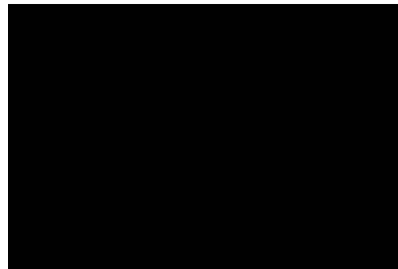
```

8 |     times = 500
9 |     error=0.001
10 | Laplace(x_upper , x_down , l_left , l_right , filas , cols , times , error)

```

## 1.3 Resultados

Los resultados son los siguientes: En la anterior imagen se puede ver, los puntos mas



calientes, donde la función alcanza sus valores máximos(color rojizo anaranjado), y la parte negra representa los valores mínimos de la función.

## 2 Problema 2

Use y modifique el código del ejercicio 1 y calcule las aproximaciones de la función  $u(x, y)$  en el cuadrado cuadrado definido por  $0 < x < 4$  y  $0 < y < 4$ , con las condiciones en la frontera.

$$u(x, 0) = 120 \text{ y } u(x, 4) = 0 \text{ para todo } 0 < x < 4 \quad u(0, y) = 120 \text{ y } u(4, y) = 0 \\ \text{para todo } 0 < y < 4$$

### 2.1 Análisis

En la siguiente tabla se especifica que valores se tomaron para el problema 2, a partir de la implementación del problema 1:

### 2.2 Programación

La implementación es la misma que el anterior problema, la diferencia radica en:

```

1 | def main():
2 |     filas = 100
3 |     cols = 100
4 |     x_upper = 0
5 |     x_down = 80
6 |     l_left = 20

```

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación  
Aprobación: 2022/03/01 Código: GUIA-PRLE-001

Parámetros:	Descripción
xu:	Será $u(x, 4) = 0 = xu$ arriba del cuadrado
xd:	Será la recta $u(x, 0) = 120 = xd$ abajo del cuadrado
yl:	Será la recta $u(0, y) = 120 = yl$ izquierda del cuadrado
yr:	Será la recta $u(4, y) = 0 = yr$ derecha del cuadrado
f:	100 filas para la matriz
c:	100 columnas para la matriz
h:	1000 iteraciones de la matriz

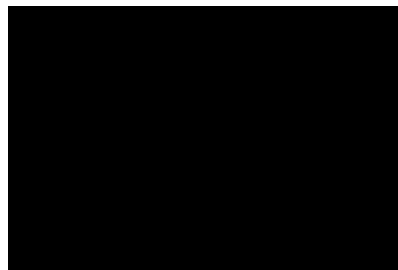
```

7 | l_right= 300
8 | times = 500
9 | error=0.001
10| Laplace(x_upper, x_down, l_left, l_right, filas, cols, times, error)

```

## 2.3 Resultados

El resultado es el siguiente:



Como se puede apreciar en la anterior figura,

## 3 Problema 3

Implementar un código computacional para la solución del siguiente potencial:

$$V(x, y) = \frac{4}{\pi} \sum_{n=1,3,5,\dots}^{\infty} \frac{1}{n} \left( \frac{V_2 \sinh(\frac{n\pi}{b}x) + V_1 \sinh(\frac{n\pi}{b}(a-x))}{\sinh(\frac{n\pi}{b}a)} \right) \sin(\frac{n\pi}{b}y) \quad (3)$$

### 3.1 Análisis

La anterior función es la solución de una ecuación diferencial, debido a esto, debemos implementar gráficamente la función pero analíticamente (debido a que la función es

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

análitica).

Para lograr esto, debemos tener en cuenta los parámetros de la función:

Variables y constantes	Descripción.
$V_1$	Potencial 1.
$V_2$	Potencial 2.
$a$	Distancia entre las dos laminas.
$b$	Anchura de las dos láminas.
$\sinh()$	Se refiere a la funcion hiperbólica de $\sin()$ .
$x,y$	son los parámetros de la función en el plano cartesiano.

### 3.2 Programación

Para la implementación del algoritmo se siguió de la siguiente manera:

1. Se creó la matriz de  $m \times n$ .
2. Luego se le indicó los límites del plano cartesiano  $(x,y)$ , en los que actuaría la función.
3. Después, se recorrió la matriz en toda su extensión con dos *for* anidados (uno para fila y el otro para las columnas).
4. Mientras se hacía el paso anterior, se puso adentro de los dos *for*, se usó la función analítica(1) a implementar con los valores  $(x,y)$  del plano cartesiano.
5. Todos estos valores se guardaban respectivamente en la matriz.
6. Luego se puso la matriz en un mapa de calor (se usó la librería *matplotlib*).

A continuación, se muestra la implementación:

```

1 from math import pi, sinh, sin
2 import numpy as np
3 import matplotlib.pyplot as plt
4 INFINITO = 100
5
6 #constantes
7 cts = {
8     "V2": 0,
9     "V1": 1,
10    "a": 1,
11    "b": 1,
12 }
13 #z=
14 # z+4*(V2*sinh((N)*pi*x/b)+V1*sinh((N)*pi*(a-x)/b)).*
```

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

```
15 # sin((N)*pi*y/b)/(sinh((N)*pi*a/b)*(N)*pi)
16
17 def funcion(x,y):
18     const = 4/pi
19     a = cts["a"]
20     b = cts["b"]
21     V1 = cts["V1"]
22     V2 = cts["V2"]
23     general2=0
24     for n in range(1,INFINITO,2):
25         # print(f"((n:{n}*pi)/b:{b})*(a:{a}-x:{x})")
26         c1 = 1/n
27         part1 = V2*sinh(((n*pi)/b)*x)
28         part2 = V1*sinh(((n*pi)/b)*(a-x))
29         denomi = sinh(((n*pi)/b)*a)
30         part3 = sin(((n*pi)/b)*y)
31         general2 += c1*((part1+part2)/denomi)*part3
32     return const*general2
33
34 def fillMatrix(arr,xu,xd,yl,yr):
35     arr[0,:] = xu
36     arr[arr.shape[0]-1,:] = xd
37     arr[0,0] = yl
38     arr[:,arr.shape[1]-1] = yr
39     arr[1:,arr.shape[1]-2]
40     #(xu+xd+yl+yr)/4
41     arr[1:arr.shape[0]-1,1:arr.shape[1]-1] = 0
42
43 def calculate(arr):
44     a = cts["a"]
45     b = cts["b"]
46
47     x=np.linspace(0,a,arr.shape[0])
48     y=np.linspace(0,b,arr.shape[1])
49
50     for i in range(0,arr.shape[0]-1):
51         for j in range(0,arr.shape[1]-1):
52             arr[i][j] = funcion(x[i],x[j])
53
54 def getCalorMap(arr):
55     plt.imshow(arr,cmap='hot',interpolation='nearest')
56     plt.show()
57
58 def analiticalFuncion(nx,ny):
59     malla = np.zeros((nx,ny))
60     # fillMatrix(malla,xu,0,0,yr)
61     calculate(malla)
62     getCalorMap(malla)
63
64 def main():
```



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01 Código: GUIA-PRLE-001

```
65     filas = 50
66     cols = 50
67     analiticalFuncion(filas , cols)
68
69
70 if __name__ == "__main__":
71     main()
```

### 3.3 Resultados

Los resultados para la función analítica son:

