

Ecuación de Calor

Viviana Niño Celis

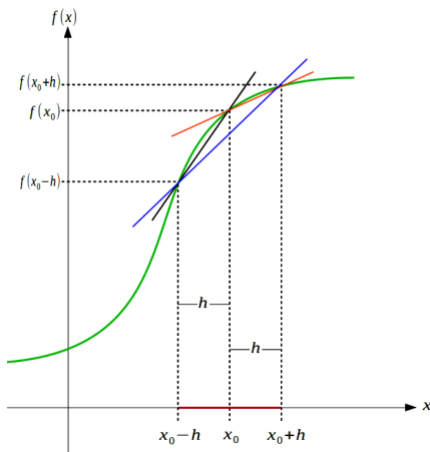
Universidad Industrial de Santander

12 de junio de 2019

Ecuación de calor

$$u_t - c\Delta u = f$$

El método numérico utilizado para la solución es diferencias finitas.



$$f(x_0) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(x_0)(x-x_0)^2 + \dots + \frac{1}{n!}f^{(n)}(x_0)(x-x_0)^n + \dots \quad (1)$$

$$f'(x_0 + h) \approx f(x_0) + f'(x_0)h \quad (2)$$

$$f'(x_0 + h) \approx f(x_0) - f'(x_0)h \quad (3)$$

$$f(x_0) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(x_0)(x-x_0)^2 + \dots + \frac{1}{n!}f^{(n)}(x_0)(x-x_0)^n + \dots \quad (1)$$

$$f'(x_0 + h) \approx f(x_0) + f'(x_0)h \quad (2)$$

$$f'(x_0 + h) \approx f(x_0) - f'(x_0)h \quad (3)$$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} \quad (4)$$

$$u_t = c\Delta u \quad (5)$$

$$u_t = c\Delta u \quad (5)$$

$$u_{i,j}^{n+1} = u_{i,j}^n + \lambda(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n) + \lambda(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) \quad (6)$$

```

def iteration1(T):
    tmp = np.zeros(T.shape)

    # Diferencias finitas
    for i in range(1, T.shape[0]-1):
        for j in range(1, T.shape[1]-1):
            #  $T[i, j] = T[i, j] + (T[i, j-1] + T[i-1, j] - 4 * T[i, j] + T[i+1, j] + T[i, j+1])$ 
            tmp[i, j] = T[i, j] + 0.25 * (T[i, j-1] + T[i-1, j] - 4 * T[i, j] + T[i+1, j] + T[i, j+1])
    return tmp

```


Constante c

Para que la ecuación diferencial y su solución tenga estabilidad y convergencia se tener en cuenta lo siguiente:

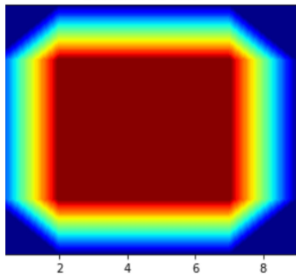
Si $\lambda \leq \frac{1}{2}$ se observa que los valores de los errores de la solución no crecen, estos tienden a oscilar.

si $\lambda \leq \frac{1}{4}$ se observa que los errores de la solución no oscilan.

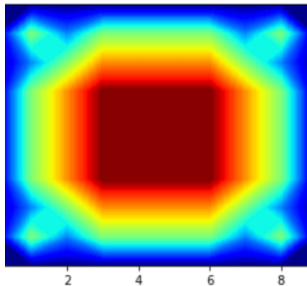
si $\lambda \leq \frac{1}{6}$ tiende a minimizar los errores por truncamiento.

$$\lambda = 1/2$$

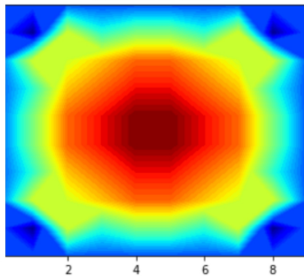
Ecuación de calor



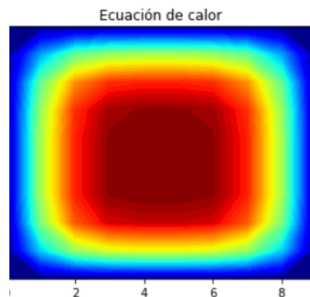
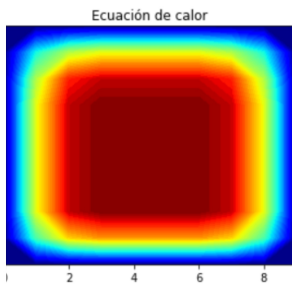
Ecuación de calor



Ecuación de calor

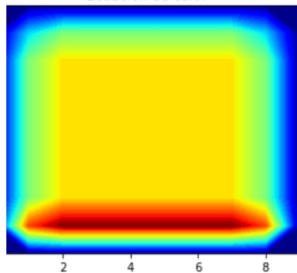


Para condición inicial 100 y valores de frontera (0,0,0,0)

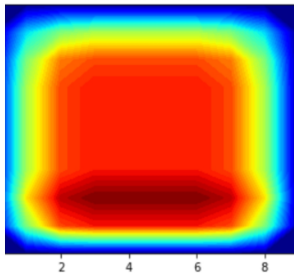


Para condición inicial 100 y valores de frontera (300,0,0,0)

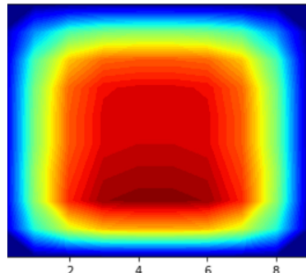
Ecuación de calor



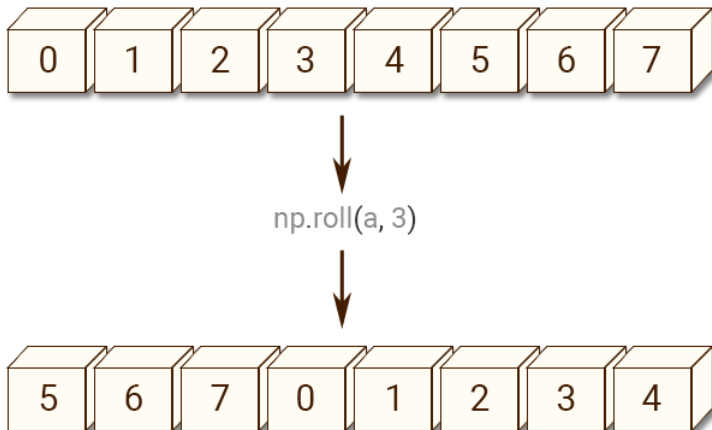
Ecuación de calor



Ecuación de calor



np.roll



© w3resource.com

Código con np.roll

```
def iteration(T):  
    tmp = np.zeros(T.shape)  
    #T[i, j] = T[i,j]+(T[i,j-1]+T[i-1,j]-4*T[i,j]+T[i+1,j]+T[i,j+1])  
    tmp=T+0.25*(np.roll(T,1,axis=0)+np.roll(T,-1,axis=0)+np.roll(T,1,axis=1)+np.roll(T,-1,axis=1))  
    return tmp
```

Tiempo

ITERACIÓN	CON ROLL	SIN ROLL
3	0.946	0.765
10	1.452	1.498
20	1.982	2.22
30	2.50	2.74
40	2.79	3.00

Utilizando bash

```
def guardaT(j,T0):  
    P=T0  
    for i in range(j):  
        if i>=0:  
            P=iteration(P)  
    return P  
  
for i in range(4):  
    Te = condiciones(5,5, rand.randrange(100,500),rand.randrange(10,100),rand.randrange(100,500),rand  
    Tem= guardaT(3,Te)
```


Utilizando Bash

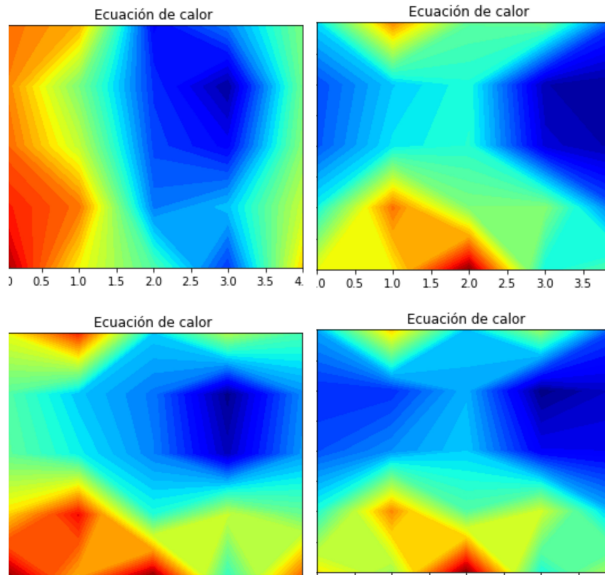
174.1	154.3	142.1	125.9	156.4
168.6	162.7	129.9	134.3	150.9
164.1	150.7	126.9	122.3	146.4
162.1	145.3	125.6	116.9	144.4
163.2	158.6	122.5	130.3	145.5

120.0	120.7	133.0	110.4	113.6
110.4	125.8	115.3	115.6	104.0
104.4	109.8	111.3	99.56	98.0
104.0	108.7	111.0	98.44	97.58
109.3	125.0	113.7	114.7	102.9

201.8	182.5	201.0	160.1	187.8
178.8	195.2	158.3	172.9	164.8
164.3	156.6	148.7	134.2	150.3
163.2	153.5	147.9	131.1	149.2
175.7	192.9	154.0	170.5	161.7

168.8	156.9	200.5	141.3	159.0
141.9	178.7	148.1	163.1	132.2
123.7	130.0	135.9	114.3	113.9
120.0	120.3	133.5	104.7	110.3
132.3	171.5	134.8	155.9	122.5

Utilizando Bash



Utilizando Clases

```
class Mapa(object):

    def __init__(self,nx,ny,Ttop,Tbottom,Tleft,Tright,Tini):
        self.x = nx
        self.y = ny
        self.Tt = Ttop
        self.Tb = Tbottom
        self.Tl = Tleft
        self.Tr = Tright
        self.Ti = Tini

        #Para llenar mi matriz de ceros inicialmente
        self.T = np.zeros((nx,ny))
        #El fill rellena la matriz con un valor, en este caso será la temperatura inicial llamada (Tini)
        self.T.fill(Tini)
        #print(T)
        # Condiciones de frontera
        self.T[0, :] = Ttop
        self.T[-1, :] = Tbottom
        self.T[:, -1] = Tright
        self.T[:, 0] = Tleft

    def iteration(self,x):
        for i in range(x):
            tmp = np.zeros(self.T.shape)
            tmp=self.T+0.25*(np.roll(self.T,1,axis=0)+np.roll(self.T,-1,axis=0)+np.roll(self.T,1,axis=1)+np.roll(self.T,-1,axis=1))
            return tmp
```