Métodos Numéricos

Fabian Trigo F. – Julio 2021

Utiliza el lenguaje Python3 para resolver la ecuacion de calor en 2 dimensiones, utilizando diferenciacion finita. Con una introduccion a la serie de Fourier, Big-O-Notation, errores en el almacenaje de numeros binarios. La simulacion fisica ocurre sin problemas y se adjunta el codigo de Python utilizado junto a capturas de distintas iteraciones en el final del documento

Introducción

Capítulo 1 Representación de Serie

Para comenzar a hablar de métodos numéricos, es necesario mejorar nuestros métodos para describe funciones, para ello se hará pivote en un pilar del análisis matemático: El Polinomio de Taylor

Esta es una aproximación a funciones en un rango dado; es una sumatoria infinita de términos, donde cada nuevo termino en la suma ejecuta una aproximación más detallada que la anterior.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \quad (1)$$

La serie de Taylor es una suma de derivadas de la función para construir la propia función alrededor del punto $x = a$.

Es impráctico el tener infinitos términos si sucesivamente se vuelven más diminutos, notase la factorial que reduce considerablemente el tamaño de términos de potencias altas. Esta serie debe truncarse; por supuesto esto viene de la mano con una truncación en el detallado de la función, es posible calcular el máximo de este error. Si expresamos la función de Taylor hasta un numero N

$$f(x) = \sum_{n=0}^{N} \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(x) \qquad (2)$$

Donde el factor R englobe todo el error de la truncación

$$R_n(x) = \frac{f^{(N+1)}(\xi)}{(N+1)!}(x-a)^{N+1} = O(x^N) \qquad (3)$$

Donde ξ es el punto en que el error R_n se maximiza, en computación podemos reducir este término por el comportamiento de x^n, aquello se llama Big-O-Notation

Capítulo 2: Discretización y el por qué la necesidad

El cálculo diferencial, ocurre cuando reducimos el grano de nuestra escala a 0, ósea utilizando el concepto de límite para pasar de una idea discreta como el cálculo de una pendiente, a una diferencial continua.

En las computadoras tartar con números pequeños presenta problemas, ya que una computadora posee un error innato producto de su grano mínimo

$$c = 2.99792 \times 10^8 \qquad (4)$$

Observes el numero "c", representa la velocidad de la luz, aquí 2.99792 es la mantisa y 8 es el exponente, en la norma *IEEE Standard 754 floating point,* habla sobre como representar números flotantes, estos son no enteros, ósea que poseen cantidades decimales. Para un flotante de precisión simple (32 bits)

$$c_B = 0\ 10011011\ 00011101110101011101010 \qquad (5)$$

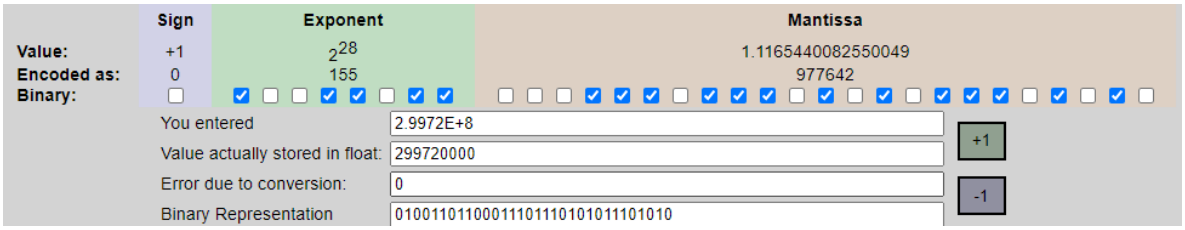

*Ilustración 1 - representación del número "c" en la norma IEEE Standard 754, la página web se adjunta en la bibliografía como (**2)*

El signo es + por tanto el primer bit es 0; los siguientes 8 bits corresponden al exponente, esto no es igual a 8, ya que se escribe el exponente en base 2. La mantisa es el decimal que en este caso es aproximadamente 1.1165…. recibe los restantes 23 bits. Este standard permite representar a un número "X":

$$1.40 \times 10^{-45} \leq || X || \leq 3.40 \times 10^{38} \qquad (6)$$

Esta representación impone un peligro, operar números muy grandes y números muy pequeños; es por esto por lo que en nuestros cálculos no podemos por ejemplo calcular una derivada con un valor de h muy pequeño *véase ecuación (7)*

$$f^{(1)}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \qquad (7)$$

Ya que, si acercamos a h a la precisión de máquina, toda precisión será perdida.

Capítulo 3 – Método de Diferencias Finitas

Como se hablo en el capitulo pasado, el calculo infinitesimal presenta un problema de precisión, es por tanto que el método de diferencias finitas es una de las soluciones.

¿Como representamos una función de varias variables? Comencemos pensando en la siguiente ecuación para tener un problema concreto en el cual pensar

$$\nabla^2 T(x,y) = \frac{\partial^2}{\partial^2 x} T + \frac{\partial^2}{\partial^2 y} T \; = \; 0$$

"T" es una función de dos variables, lo que se hace es subdividir el espacio en puntos, esto se llama discretizar; "x" e "y" ya no serán variables continuas con granulado infinitesimal. Puede observarse la división del espacio en la siguiente ilustración:

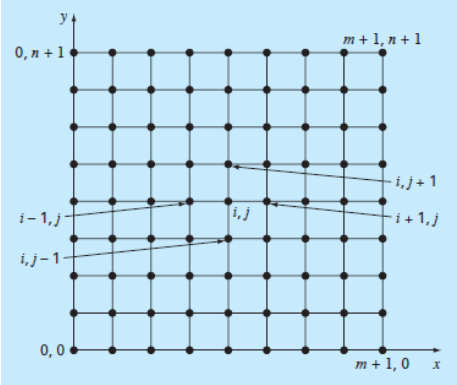

Ilustración 2 - Discretización del Espacio

El granulado "h", geométricamente la arista de una celda representada en la figura anterior, obsérvese la siguiente relación para las coordenadas "x" y el tamaño "h" de las celdas

$x_0 = 0 \; ; x_1 = h \; ; \ldots ; \; x_i = i \times h$

La coordenada "y" sigue esta misma relación, así nuestra función de forma discreta:

$$T_{i;j} = T(x_i, y_j)$$

Puede ser descrita como una matriz de valores. Las computadoras son expertas en matrices, a esto generalmente se le llama Array o Arreglo

Ahora aplicando lo del *capítulo 1*, podemos escribir la función cerca al punto $(x_i; y_j)$, la expansión de Taylor usa derivadas en "x"

$$T_{i+1;j} = T(x_i + h, y_j) = T(x_i; y_j) + T'(x_i; y_j)h + \frac{T''(x_i; y_j)}{2!}h^2 + \frac{T'''(x_i; y_j)}{3!}h^3 + \cdots$$

$$T_{i-1;j} = T(x_i - h, y_j) = T(x_i; y_j) \; - \; T'(x_i; y_j)h + \frac{T''(x_i; y_j)}{2!}h^2 - \frac{T'''(x_i; y_j)}{3!}h^3 + ..$$

Ahora restándole la 2da a la 1era expansión y despejando la primera derivada

$$T_{i+1;j} - T_{i-1;j} = 2T'(x_i; y_j)h \; + 2 \; \frac{T'''(x_i; y_j)}{3!}h^3$$

$$T'(x_i; y_j) = \frac{\partial}{\partial \, x} T_{i;j} = \frac{T_{i+1;j} - T_{i-1;j}}{2h} + O(h^2)$$

Sumando 1era y 2da expansión, obtenemos la 2da derivada. Para ahorrar espacio se expondrán las expresiones de las derivadas

$$\frac{\partial}{\partial x}T_{i;j} = \frac{T_{i+1;j} - T_{i-1;j}}{2h} + O(h^2) \;\; ; \;\; \frac{\partial}{\partial y}T_{i;j} = \frac{T_{i;j+1} - T_{i;j-1}}{2h} + O(h^2)$$

$$\frac{\partial^2}{\partial^2 x}T_{i;j} = \frac{T_{i+1;j} - 2T_{i;j} + T_{i-1;j}}{h^2} + O(h^2)$$

En el caso que $T = T(x, y, t)$ una función de 3 variables sería un arreglo con 3 dimensiones, $T(x_i, y_j, t_n) = T^n_{i;j}$ la derivada en t seria la siguiente:

$$\frac{\partial}{\partial t}T^n_{i;j} = \frac{T^{n+1}{}_{i;j} - T^{n-1}{}_{i;j}}{2h} + O(h^2)$$

En el siguiente capítulo desarrollaremos el método a partir de un ejemplo/aplicación

Capítulo 4 – Aplicación a Ecuación de Calor

Tengamos la siguiente situación, (*ilustración 3 - izquierda*) una batería de teléfono, a la cual deseamos calcular el flujo de calor a través de su superficie, para esto seguiría la ecuación de calor.

$$\rho c_p \frac{\partial}{\partial t}T - \nabla \cdot (\mathrm{k}\nabla \mathrm{T}) = Q$$

Donde T es la Temperatura $T = T(x, y, z, t)$

k es la conductividad térmica; $k = 140[W/mK]$ para este problema

ρ es la densidad del material/medio; $\rho = 2200[k/m^3]$

c_p es el calor especifico a presión constante; $c_p = 0.720[J/Kkg]$

Q es una fuente externa que introduce energía uniformemente (esta ecuación no incluye transferencia de masa)

$\nabla \cdot (\mathrm{k}\nabla \mathrm{T})$ es la divergencia de la gradiente de la temperatura por la conductividad térmica

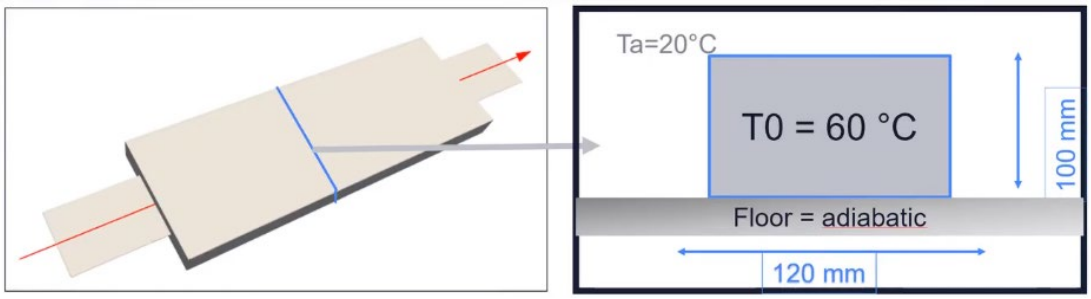

Ilustración 3 - Batería de Teléfono, paralela a línea roja: eje z. debido a la simetría en el eje z impuesta se convierte en un problema bidimensional xy

Esta batería es uniforme, no posee cambios en la conductividad térmica en el espacio

$$\nabla \cdot (k\nabla T) = \nabla k \cdot \nabla T + k\nabla^2 T = k\nabla^2 T$$

Esta es simétrica en el eje z, ósea que es idéntica en todo este largo, por tanto, podemos resolver solo una rodaja o sección bidimensional del problema de calor.

$$\nabla^2 T = \left(\frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y} + \frac{\partial^2}{\partial^2 z}\right) T \ = \frac{\partial^2}{\partial^2 x} T + \frac{\partial^2}{\partial^2 y} T$$

Esta batería se encuentra a una temperatura interna de 60 grados Celsius, mientras que el ambiente se trata como un reservorio a temperatura constante 20 grados. Que el ambiente sea un reservorio a temperatura constante quiere decir que puede fluir calor hacia este y es tan grande que no sufrirá modificaciones. Otra aclaración es que el piso es adiabático, esto quiere decir que no existen intercambios de calor

De esa forma simplificamos la ecuación de calor

$$Q \ = \ 0 \quad ; \quad \frac{\partial}{\partial t} T - \frac{k}{\rho c_p} \nabla^2 T = 0$$

Podemos reducir las constantes a un único numero

$$\frac{k}{\rho c_p} = \alpha = \frac{140}{2200 \times 0.720} \left[\frac{W}{m\ K}\right] \left[\frac{m^3}{kg}\right] \left[\frac{kg\ K}{J}\right] = \frac{7}{792} \left[\frac{m^2}{s}\right] \quad ; \quad \frac{\partial}{\partial t} T = \alpha \nabla^2 T$$

Aplicando el método visto en *capítulo 3* de diferencias finitas, las derivadas de ese capítulo serán escritas aquí.

$$\frac{T^{n+1} - T^n}{\Delta t} = \alpha \left(\frac{T_{i+1;j} - 2T_{i;j} + T_{i-1;j}}{(\Delta x)^2} + \frac{T_{i;j+1} - 2T_{i;j} + T_{i;j-1}}{(\Delta y)^2} \right); \quad \Delta x = \ \Delta y = h; \ \Delta \mathrm{t} \ = dt$$

Ahora a crear el arreglo con el que trabajaremos: Iniciamos el arreglo $T_{i;j}^0$ de la siguiente manera:

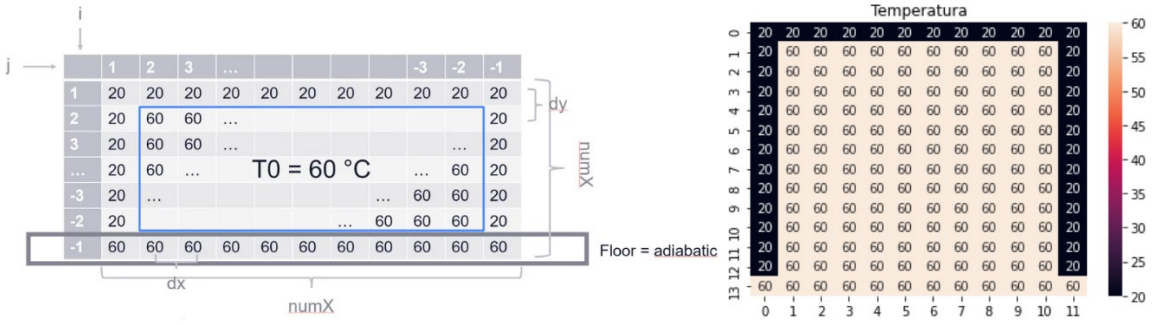

Ilustración 4 - Arreglo Tij en el instante inicial, izquierda es la teoría, derecha es la matriz que tenemos en la simulación de Python adjunta

Nótese el superíndice 0, esto quiere decir que nos encontramos en el instante inicial, nuestro trabajo es encontrar la evolución temporal, ósea a partir de un termino T^n hallar el T^{n+1}. En forma de función queremos $T^{n+1} = T^{n+1}(T^n)$.

$$T^{n+1} = {T_{i;j}}^n + \alpha \mathrm{dt}\left(\frac{T_{i+1;j} - 2T_{i;j} + T_{i-1;j}}{h^2} + \frac{T_{i;j+1} - 2T_{i;j} + T_{i;j-1}}{h^2}\right) + O(h^2\, dt)$$

Esta es *la fórmula iterativa de temperatura*, cada vez que la apliquemos actualizara los valores de la matriz, observemos el siguiente código en Python3:

```
import numpy as np #realizar calculo de matrices
import matplotlib.pyplot as plt #visualizar
import seaborn as sns

dt = 0.1 # es el paso diminuto en el tiempo
h = 1    # el granulado del espacio (para simplicidad en visualizacion)
alpha = 7/792 # alpha = k/rho*cp [m^2/s]
geo = [12 + 2, 10 + 2] #x,y dimensiones del espacio, (x;y)=12cm;10cm es la bateria
```

```
#inicializamos el arreglo
T = np.full((geo[1],geo[0]),20) #llenamos todo de 20 grados
T[-1,::] ,T[1:-1,1:-1:] = 60, 60 #dejamos 60 grados en lo que es la bateria y la parte adiabatica
plot(T) #grafica T
```

```
#en python, ya que son matrices: T_x;y = T[y,x]; esta celda aplica la iteracion
for y in range(1,11):
    for x in range(1,13): #x
        d2dy= (T[y+1, x] - 2*T[y,x] + T[y-1,x] ) #derivada 2da en x
        d2dx= (T[y, x+1] - 2*T[y,x] + T[y,x-1]) #derivada 2da en y
        T[y,x] = T[y,x] + alpha*dt*( d2dx + d2dy)
plot(T)
```

Este código es propio, por tanto, no aplica citar. El primer bloque importa librerías, el 2do inicializa el arreglo con las condiciones iniciales, el 3ro realiza la iteración en el tiempo, este ultimo se corre varias veces.

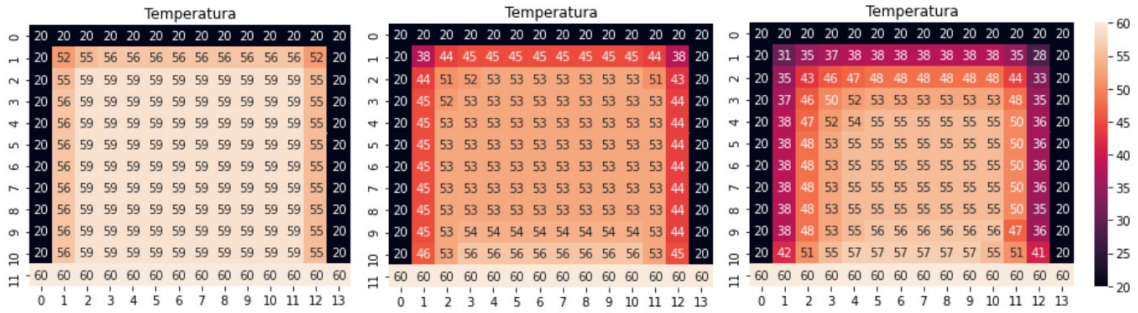

Ilustración 5 - Izquierda: Iteración 10; Centro: Iteración 35; Iteración 66

Bibliografía

(*1) Bornemann. Folkmar. y Weisstein, Eric W. "Power Series." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/PowerSeries.html

(*2) Schmidt (2017). “Float Converter Page” H-Schmidt Web Source https://www.h-schmidt.net/FloatConverter/IEEE754.html

(*3) Raymon, C. y Steven C. (2010) “Numerical Methods for Engineers” (6th ed, Vol. 1). New York. McGrawHill. ISBN 978-0-07-340106-5