FIRST MIDTERM

Nataly Phawllyn Neira Parra Cod: 614212782
Fundación Universitaria Konrad Lorenz

14 de septiembre de 2024

Computación científica II

Expansión de Taylor de un función de una variable , centrada en x_0

$$f(x_0+h)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}h^n=f(x_0)+\frac{f'(x_0)}{1!}h+\frac{f''(x_0)}{2!}h^2+\frac{f^{(3)}(x_0)}{3!}h^3\cdots \tag{1}$$

Aproximación de la primera derivada con respecto a x es:

$$f'(x_0)=\frac{f(x_0+h)-f(x_0-h)}{2h}+\mathcal{O}(h^2) \tag{2}$$

Ejercicio. 1 Encuentre una aproximación para la derivada de tercer orden de f en x_0 donde el error sea del orden de $\mathcal{O}(h^2)$

Solución: Consideremos la expansión de Taylor (Ecu.1) para $f(x_0+2h)$

$$\begin{aligned}f(x_0+(2h))&=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(2h)^n\\&=f(x_0)+\frac{f'(x_0)}{1!}2h+\frac{f''(x_0)}{2!}4h^2+\frac{f^{(3)}(x_0)}{3!}8h^3+\frac{f^{(4)}(x_0)}{4!}16h^4\cdots\end{aligned}$$

la expansión de Taylor para $f(x_0-2h)$

$$\begin{aligned}f(x_0-(2h))&=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(-2h)^n\\&=f(x_0)-\frac{f'(x_0)}{1!}2h+\frac{f''(x_0)}{2!}4h^2-\frac{f^{(3)}(x_0)}{3!}8h^3+\frac{f^{(4)}(x_0)}{4!}16h^4-\cdots\end{aligned}$$

Si restamos $f(x_0+2h)-f(x_0-2h)$ tenemos que las derivadas pares se anulan, y al despejar la tercera derivada tenemos:

$$\begin{aligned}f(x_0+2h)-f(x_0-2h)&=2\frac{f'(x_0)}{1!}2h+2\frac{f^{(3)}(x_0)}{3!}8h^3+2\frac{f^{(5)}(x_0)}{5!}(2h)^5+\cdots\\&=\frac{4hf'(x_0)}{1!}+\frac{16h^3f^{(3)}(x_0)}{3!}+\frac{2^6h^5f^{(5)}(x_0)}{5!}+\cdots\\\rightarrow\frac{16h^3f^{(3)}(x_0)}{3!}&=f(x_0+2h)-f(x_0-2h)-\frac{4hf'(x_0)}{1!}-\frac{2^6h^5f^{(5)}(x_0)}{5!}+\cdots\\\rightarrow f^{(3)}(x_0)&=\frac{3!}{16h^3}\left(f(x_0+2h)-f(x_0-2h)-\frac{4hf'(x_0)}{1!}-\frac{2^6h^5f^{(5)}(x_0)}{5!}+\cdots\right)\\&=\frac{6f(x_0+2h)-6f(x_0-2h)}{16h^3}-\frac{24hf'(x_0)}{16h^3}-\frac{3!2^6h^5f^{(5)}(x_0)}{5!16h^3}+\cdots\\&=\frac{3f(x_0+2h)-3f(x_0-2h)}{8h^3}-\frac{3f'(x_0)}{2h^2}-\frac{h^2f^{(5)}(x_0)}{5}+\cdots\end{aligned}$$

Usamos la Ecu.2 para reemplazar la primera derivada.

$$\begin{aligned} f^{(3)}(x_0) &= \frac{3f(x_0+2h)-3f(x_0-2h)}{8h^3} - \frac{3}{2h^2}\left(\frac{f(x_0+h)-f(x_0-h)}{2h} + \mathcal{O}(h^2)\right) - \xi h^2 + \cdots \\ &= \frac{3f(x_0+2h)-3f(x_0-2h)}{8h^3} + \frac{3f(x_0-h)-3f(x_0+h)}{4h^3} - \xi' - \xi h^2 + \cdots \end{aligned}$$

Tenemos entonces la aproximación para la derivada de tercer orden como

$$f^{(3)}(x_0) = \frac{3f(x_0+2h)-3f(x_0-2h)+6f(x_0-h)-6f(x_0+h)}{8h^3} - \mathcal{O}(h^2) \tag{3}$$

Ejercicio. 2 De un método numérico basado en las diferencias finitas para integrar numéricamente la siguiente ecuación diferencial

$$\frac{\partial \phi}{\partial t} + \frac{\partial^3 \phi}{\partial x^3} + \phi \frac{\partial \phi}{\partial x} = 0 \tag{4}$$

$x \in \mathbf{R}, t \geq 0$. No olvide reportar el orden de Convergencia del método.

Solución: El método numérico esta implementado en la Función **KdV_solver** del archivo python adjunto (**parcial1_1_neira.py**). Durante todo el scrip se intenta evitar el uso de *for's* y se intenta vectorizar todo proceso que se pueda ,además del uso de la función *ne.evaluate*,

Para la implementación de este método , se discretiza el tiempo y el espacio tal que

$$0 = t_0 < t_1 < \cdots < t_J = T_{final}$$

$$0 = x_0 < x_1 < \cdots < x_N = L$$

con una diferencia fija de $t_{j+1} - t_j = s$ y $x_{n+1} - x_n = h$Aunque el domino de la función en su componente espacial es infinito, se considera una malla discreta para poder ser manejado computacional mente, esta limitación debe compensarse con el uso adecuado de condiciones de frontera.

Escribimos la EDP.4 haciendo uso de las aproximaciones encontradas en Ecu.2 y Ecu.3, donde tenemos $\phi(x_n, t_j) = \phi_n^j$

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= -\frac{\partial^3 \phi}{\partial x^3} - \phi \frac{\partial \phi}{\partial x} \\ \frac{\phi_n^{j+1} - \phi_n^{j-1}}{2s} &= -\frac{3\phi_{n+2}^j - 3\phi_{n-2}^j + 6\phi_{n-1}^j - 6\phi_{n+1}^j}{8h^3} - \phi_n^j \left(\frac{\phi_{n+1}^j - \phi_{n-1}^j}{2h}\right) \end{aligned}$$

de donde podemos tener una ecuación recurrente para el tiempo en los nodos centrales

$$\phi_n^{j+1} = \phi_n^{j-1} + \frac{3s}{4h^3}\left(\phi_{n-2}^j - \phi_{n+2}^j - 2\phi_{n-1}^j + 2\phi_{n+1}^j\right) + \frac{s}{h}\phi_n^j\left(\phi_{n-1}^j - \phi_{n+1}^j\right) \tag{5}$$

Veamos que para calcular ϕ_n^{j+1} necesitan las dos arreglos espaciales anteriores ϕ^j y ϕ^{j-1}. por lo tanto, para los nodos centrales con $j \geq 1$ es necesario conocer ϕ^0 y ϕ^1, donde ϕ^0 que dependerá de las condición iniciales.

Por otra parte, para ϕ^1 realizaremos una aproximación al valor usando diferencias finitas hacia adelante para la derivada en el tiempo en $t = 0$, teniendo así

$$\phi_n^1 = \phi_n^0 + \frac{3s}{8h^3}\left(\phi_{n-2}^0 - \phi_{n+2}^0 - 2\phi_{n-1}^0 + 2\phi_{n+1}^0\right) + \frac{s}{2h}\phi_n^0\left(\phi_{n-1}^0 - \phi_{n+1}^0\right)$$

Observación: no se usa diferencias finitas hacia adelante en los nodos centrales con la esperanza de que no altere el orden de convergencia pues, la diferencia finitas centradas son de orden $\mathcal{O}(s^2)$ y las diferencias finitas hacia adelante (o hacia atrás) son de orden $\mathcal{O}(s)$

CONVERGENCIA DEL MÉTODO

La convergencia de este método está sujeto a de criterios de convergencia de tipo *CFL (Courant-Friedrichs-Lewy)* el cual nos indica que para la ecuación de KdV , el metodo converge si :

$$\frac{3s}{4h^3} < C_1$$
$$\frac{s}{h} < C_2$$

donde C_1 y C_2 son constantes que deben estar entre cero y 1 simultáneamente.Por lo tanto la elección del tamaño del paso temporal y espacial no puede ser aleatoria.

por una parte, C_1 cumple el criterio si $\frac{s}{h} < 1$, por lo tanto $s < h$. Por otra parte tenemos que :

$$\frac{3}{4} < 1$$
$$\frac{3s}{4h^3} < \frac{s}{h^3}$$
$$C_1 < \frac{s}{h^3}$$

así que C_2 cumple el criterio si $\frac{s}{h^3} < 1$, por lo tanto $s < h^3$. y como pedimos pasos de $h < 1$ pequeños, tenemos que $h^3 < h$, entonces $s < h^3 < h$ así ambas condiciones se cumplen si $s < h^3$ en cada par de pasos elegidos para la rejilla.

observación: Se usa la solución encontrada con la malla mas fina como solución "exacta" para probar la convergencia del método.

Ejercicio. 3 Integre numéricamente la la EDP.4 en python en un intervalo finito, usando condiciones iniciales $\phi(x, o)$ significativas. De su solución como una simulación donde se vea la evolución de la onda en el tiempo.

Solución: Como se explica en el punto anterior se usara un intervalo finito $0 \geq x \geq L$ y $0 \geq t \geq T_{final}$.

CONDICIONES DE FRONTERA

Para este caso se usaran condiciones de frontera periódicas pues la KdV se usa usualmente para describir las olas en aguas poco profundas, lo cual no se limita estrictamente a un intervalo finito, pero si presenta comportamientos periódicos.

$$\phi(0,t) = \phi(L,t)$$
$$\frac{\partial\phi}{\partial x}(0,t) = \frac{\partial\phi}{\partial x}(L,t)$$
$$\frac{\partial^2\phi}{\partial x^2}(0,t) = \frac{\partial^2\phi}{\partial x^2}(L,t)$$

CONDICIONES INICIALES

Para la condición inicial tomaremos una forma sinusoidal con periodo L y amplitud 0.5 de la forma

$$\phi(x,0) = \frac{1}{2} sen\left(\frac{2\pi}{L}x\right) + \frac{1}{2}$$

Esta condición inicial, nos asegura que $\phi(x,t)$ esta entre 0 y 1. a iniciar.

Para simular la solución se uso un $L = 10$ y un $T_{final} = 10$ y $h = 10^{-1}$ Fig.1. de la cual se puede ver es una función continua en todo el tiempo. El .gif se encuentra en los archivos enviados

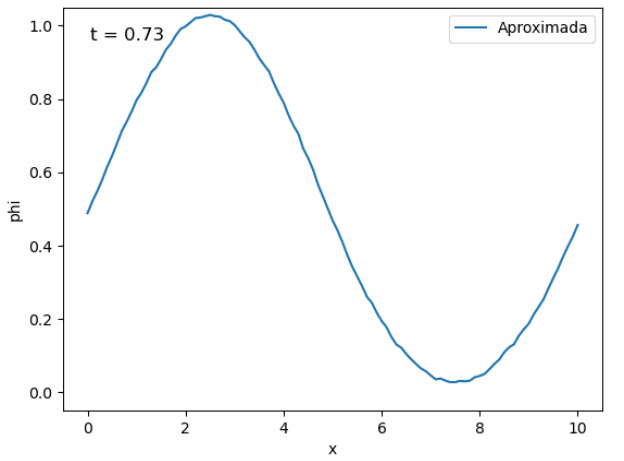

Figura 1: simulación de la Onda

Ejercicio. 4
PARALLEL SORTING:
Tome un arreglo de números enteros aleatorios con distribución uniforme de $0 - 100$ de tamaño $N = 10^6$ y organícelos usando su implementación en paralelo en python. Compare su desempeño con la implementación secuencial.

Solución:
SECUENCIAL:
se crea una función **sort_array_min** en el archivo **sort_seq.py** que organiza el arreglo haciendo uso de la función de mínimos de numpy, esta programado secuencialmente.
Podemos ver que para tamaños muy grandes el tiempo de ejecución no es óptimo, por ejemplo para un arreglo de tamaño $N = 10^5$ tarda más de 2 minutos (Fig. 2) lo cual no lo hace apto para organizar arreglos del tamaños mas grandes

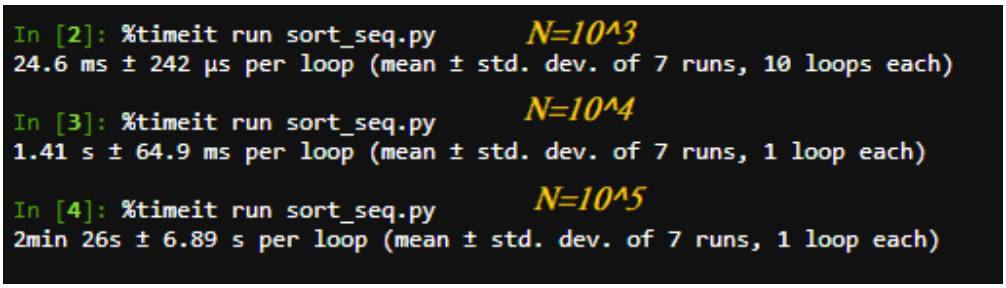

Figura 2: tiempo de ejecución en secuencial

PARALELO
La función principal **sort_paralelo** en el archivo **parcial1_2_neira** divide en 4 el arreglo original, según unos rangos de valores y pone a 4 procesadores a ordenar cada parte, para luego ser concatenada. Como la distribución de datos aleatorios es uniforme se espera que la carga sobre cada procesador también lo sea.
Para organizar cada arreglos se uso la función **sort_div** que usa la división del arreglo recursivamente, en arreglos con cierto rango de datos, ordenando implícitamente. La función es hecha pensada bajo la lógica de ser usada en paralelo, pues así no excede la profundidad máxima de recursividad del computador, algo que si sucedía al intentar usar la función creada en secuencial(*sort_array_min(arr)*), para arreglos mas pequeños.

Se obtiene para un arreglo de $N = 10^6$ un tiempo de ejecución de aproximadamente 2 segundos. esta

mejoría se debe: primero a la paralelización del código y segundo al cambio de la función que organiza el arreglo.

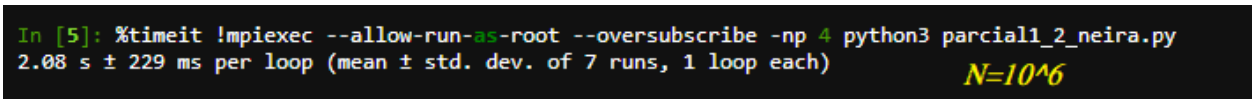

Figura 3: tiempo de ejecución en paralelo

Nota: Los Scripts donde se solucionan los puntos se encuentran adjuntas, en archivo .zip donde se encontraba este documento.

.