Física Computacional (F811)

Primer Proyecto

Guatemala, 5 de marzo de 2021

Instrucciones: Presentar un informe completo sobre el desarrollo y la implementación en Fortran 95 de su solución a cada problema. Preparar un único archivo tar.gz para adjuntar todos los archivos usados para elaborar el informe: un capítulo del informe por cada problema y los códigos fuente necesarios para compilar todas las implementaciones. El requisito para calificar el problema es que el código debe compilar sin errores en el clúster euclides.

1. Conjetura de Goldbach (10 puntos)

La conjetura se debe al matemático alemán Christian Goldbach quien en 1742, escribió en una carta a Euler una conjetura

> *Cada entero que puede escribirse como la suma de dos primos, también puede escribirse como la suma de tantos primos como uno desee, hasta que todos los términos sean la unidad.*

en esa misma carta también escribió al margen una segunda conjetura: *cada entero mayor que 2 puede escribirse como la suma de tres primos*. Hay que considerar que en aquella época todavía se consideraba que el número uno era primo por lo que en la actualidad, la Conjetura de Goldbach se enuncia como *cada entero mayor que 5 puede escribirse como la suma de tres primos* o como *cada entero par mayor que 2 puede escribirse como la suma de dos primos*. Ahora también se sabe que ambas versiones son equivalentes.

1. Implemente un código en Fortran para que encuentre los números primos menores que un número n dado. Por ejemplo, para $n = 10$, los números primos son $\{2, 3, 5, 7\}$.

2. Implemente un código en Fortran para probar que la conjetura de Goldbach se cumple. Por ejemplo: para $n = 10$, se tiene $10 = (3 + 7) = (5 + 5) = (3 + 5 + 2)$.

3. Uno de los problemas asociados a esta conjetura es el de buscar de cuántas formas se puede representar un número n. Siguiendo el ejemplo de $n = 10$, existen dos formas usando dos primos, $v_{n,2} = 2$ y sólo una usando tres primos $v_{n,3} = 1$. Muestre una gráfica con el comportamiento de $v_{n,2}$ y $v_{n,3}$.

4. ¿Es posible escribir una implementación en paralelo? (Escriba los argumentos de cómo dividir las tareas o los datos).

5. Si la respuesta anterior es afirmativa, implemente un código en Fortran siguiendo los argumentos presentados previamente.

6. Si hizo el inciso anterior, use el tiempo de cómputo total para estimar la complejidad de este problema.

2. Aguja de Buffon (10 puntos)

El problema de la aguja de Buffon fue propuesto por Georges-Louis Leclerc, Conde de Buffon, en uno de sus libros de 1733 y consiste en encontrar la probabilidad de que una aguja, de longitud ℓ, que se lanza al suelo donde se han dibujado líneas paralelas, separadas una distancia d, toque una de esas líneas.

El problema también fue resuelto por Buffon en 1777 definiendo un parámetro de distancia

$$x \equiv \frac{\ell}{d},$$

de modo que para una aguja pequeña, $x < 1$, la probabilidad de que la aguja toque una línea es

$$\begin{aligned} P(x) &= \int_0^{2\pi} \frac{\ell|\cos(\theta)|}{2\pi d} d\theta \\ &= \frac{2\ell}{\pi d} \int_0^{\pi/2} \cos(\theta) d\theta \\ &= \frac{2\ell}{\pi d} \\ &= \frac{2x}{\pi}. \end{aligned}$$

Para una aguja de tamaño $x = 1$, se tiene $P(1) = 2/\pi$, que resulta en una forma entretenida de calcular decimales para π.

Para una aguja más grande, $x > 1$, el cálculo de la función de probabilidad es más complejo pero se sabe que

$$P(x) = \frac{2}{\pi}\left(x - \sqrt{x^2 - 1} + \sec^{-1}(x)\right).$$

Volviendo al caso $x = 1$, podemos hacer una simulación para estimar $P(1)$ de modo que podemos aproximar $\pi \approx 2/P(1)$. La simulación involucra soltar N agujas sobre el suelo y que n de ellas crucen las líneas paralelas de modo que podemos aproximar $P(1) \approx n/N$.

1. Escriba un programa en Fortran que haga la simulación para $x = 1$ de modo que podamos obtener aproximaciones de π.

2. Muestre el comportamiento del valor aproximado de π en función del número de agujas N. ¿Cuál es el comportamiento del error absoluto en función del número de agujas N?

3. Haga simulaciones usando (a) precisión sencilla, (b) doble precisión y (c) cuádruple precisión.

4. ¿Es posible escribir una implementación en paralelo? Argumente su respuesta diciendo cómo se dividirían las tareas o los datos.

5. Si su respuesta anterior es afirmativa, ¿hay una mejora en el tiempo total de cómputo utilizado para la simulación?

3. Funciones de densidad de probabilidad (10 puntos)

La función de densidad de probabilidad de una variable aleatoria, o sólo función de densidad, describe la probabilidad relativa según la cual dicha variable tomará un valor determinado. Los generadores de números pseudoaleatorios más comunes producen números siguiendo una distribución uniforme, es decir, todos los números tienen la misma probabilidad de aparecer.

Si queremos modificar la función de densidad de una distribución podemos usar varios métodos por ejemplo, ya usamos antes el *método de suma de funciones uniformes*, que genera una función de densidad de probabilidad gaussiana.

También se puede usar el *método inverso* donde consideramos que u_i es una variable aleatoria con una función de densidad de probabilidad uniforme $U(u)$, entonces una variable aleatoria t_i que sigue una función de densidad de probabilidad normal, $G(t)$, está dada por la transformación

$$t_i = \left(\ln \frac{1}{u_i^2} \right)^{1/2}.$$

Para generar una distribución exponencial, $F(x) = e^{-\delta x}$, usando una distribución uniforme también podemos usar el método inverso, usando la transformación

$$x_i = -\frac{1}{\delta} \ln(u_i),$$

donde $\delta \neq 0$ es un parámetro que determina *qué tan desordenada* es la distribución.

a) Implemente un código de Fortran que genere un conjunto de números aleatorios $\{u_i\}$ y que luego los transforme usando el método inverso a

1) una distribución normal,

2) una distribución exponencial.

b) ¿Es posible hacer implementaciones en paralelo? (Escriba los argumentos de cómo dividir las tareas o los datos).

c) Si la respuesta anterior es afirmativa. Implemente los códigos en Fortran para hacer las transformaciones del inciso (a).

d) Si hizo el inciso anterior, use el tiempo de cómputo total para estimar la complejidad de transformar una distribución uniforme a una distribución exponencial. ¿Cómo afecta el parámetro δ a esta complejidad?

Sugerencias: puede usar histogramas de frecuencias para comprobar las transformaciones.

4. Números de Fibonacci (10 puntos)

La secuencia de Fibonacci fue introducida por Leonardo Bonacci (también llamado Leonardo de Pisa o Leonardo Bigollo Pisano o Fibonacci) en 1202 con una explicación usando parejas de conejos: se supone que una pareja de conejos tendrá un par de gazapos cada mes y que a los dos meses de vida los conejos serán capaces de reproducirse, en esta analogía los conejos no mueren. Si al inicio se compra una pareja de gazapos, al final del primero y segundo mes se tendrá sólo una pareja, pero al final del tercer mes se tendrán dos parejas. En el cuarto mes se tendrán tres parejas y en el quinto mes habrán cinco parejas y así se obtiene la secuencia de Fibonacci $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$. Matemáticamente, el n-ésimo número de la secuencia de Fibonacci se define como

$$f_n = f_{n-1} + f_{n-2},$$

para $n \geq 2$, con $f_0 \equiv 0$ y $f_1 \equiv 1$. Otra forma de obtener los números de la secuencia es con la multiplicación de matrices

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix},$$

de donde se pueden tomar el $(n+1)$-ésimo número de la secuencia en n pasos.

a) Implemente un código de Fortran para calcular el n-ésimo número de la secuencia de Fibonacci mediante (a) la relación $f_n = f_{n-1} + f_{n-2}$, y también (b) usando la forma matricial.

b) Usando el tiempo total de cómputo y la memoria RAM consumida como parámetros de rendimiento del algoritmo. ¿Cuál implementación es más favorable?

c) ¿Es posible plantear una solución en paralelo? (Escriba los argumentos de cómo dividir las tareas o los datos)

d) Si la respuesta anterior es afirmativa. Implemente una versión en paralelo para calcular en n-ésimo número de la secuencia.

e) Si hizo el inciso anterior, ¿cómo cambia el comportamiento del tiempo total de cómputo?

5. Función Gamma (10 puntos)

La función gamma, $\Gamma(z)$, aparece en algunos problemas de física, por ejemplo en algunos cálculos de probabilidades en mecánica estadística. Existen varias formas de definir la función gamma, por ejemplo con el límite infinito, también llamado de Euler

$$\Gamma(z) \equiv \lim_{n\to\infty} \frac{1\cdot 2\cdot 3\cdots n}{z(z+1)(z+2)\cdots(z+n)} n^z,$$

para $z \neq 0, -1, -2, -3, \ldots$, de donde se puede obtener la forma de producto infinito de Weierstraß

$$\Gamma(z+1) = \lim_{n\to\infty} \left(\frac{nz}{z+n+1}\right) \frac{1\cdot 2\cdot 3\cdots n}{z(z+1)(z+2)\cdots(z+n)} n^z,$$

para encontrar la propiedad $\Gamma(z+1) = z\Gamma(z)$. Una segunda definición, también llamada integral de Euler es

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt,$$

donde $z \in \mathbb{C}$, sujeta a $\Re(z) > 0$ de modo que la integral sea convergente.

a) Implemente un código de Fortran para encontrar la función gamma. Elija la definición que desee.

b) ¿Es posible plantear una solución recursiva? (Escribir argumentos).

c) Si la respuesta anterior es afirmativa. Implemente una versión recursiva y compare su rendimiento con la opción secuencial.

Sugerencias: si utiliza la fórmula del producto infinito inicie con números n pequeños, pruebe la convergencia de su solución para la precisión con la que está trabajando.

6. Torres de Hanói (10 puntos)

Las Torres de Hanói están compuestas por una serie de discos de diferente radio que están puestas en orden de decreciente de abajo hacia arriba en uno de los tres postes verticales que se usan en el juego. El juego consiste en pasar todos los discos a otro de los postes siguiendo un conjunto de reglas: (1) solo se puede mover un disco cada vez y para mover otro, los demás tienen que estar en postes, (2) un disco de mayor tamaño no puede estar nunca sobre uno más pequeño, y (3) solo se puede desplazar el disco que se encuentre arriba en cada poste.

a) Implemente un código recursivo de Fortran que cuente los pasos necesarios para n discos. Imprima el número de instrucción y la instrucción, por ejemplo: *mover disco 1 del poste A al poste C.*

b) Compruebe que la complejidad de este problema es exponencial con n.

c) El programa debe incluir la posibilidad de imprimir los pasos que se deben seguir para pasar todos los discos a otro poste.

d) Discuta si la medida de complejidad está de acuerdo las mediciones de tiempo y memoria usados en función de n.

Sugerencia: tener cuidado con el número de discos. La complejidad del problema es $2^n - 1$. Dejar la posibilidad de pasarle un parámetro en tiempo de ejecución al programa para que imprima o no las instrucciones d.

7. Producto de Kronecker (10 puntos)

El producto de Kronecker es una operación entre matrices de tamaños arbitrarios. Considere las matrices $\bar{A}$, de dimensión $m \times n$, y $\bar{B}$, de dimensión $p \times q$, entonces el producto de Kronecker define una matriz $\bar{C} = \bar{A} \otimes \bar{B}$, de dimensión $(mp) \times (nq)$, cuyos elementos están dados por

$$c_{\alpha,\beta} = a_{ij} b_{kl} \ ,$$

donde $\alpha \equiv p(i-1) + k$ y $\beta \equiv q(j-1) + l$.

a) Implemente un código de Fortran para calcular el producto de Kronecker de dos matrices de dimensión arbitraria.

b) ¿Cuál es la complejidad de este problema? Haga las aproximaciones del tiempo y de la memoria usados en función de la dimensión de la matriz C.

c) ¿Es posible resolver el problema de forma paralela? (Escriba los argumentos de cómo dividir las tareas o los datos).

d) Si la respuesta anterior es afirmativa. Implemente una versión paralela del código.

e) Si hizo el inciso anterior. Compare el uso de recursos de su implementación secuencial y su implementación paralela.

8. Método de bisección (10 puntos)

Sea f una función continua en $[a, b]$ que satisface la *condición de cambio de signo*, $f(a)f(b) < 0$. Entonces, necesariamente f tiene por lo menos un cero en (a, b). La estrategia del método de bisección es dividir el intervalo por la mitad y elegir el subintervalo donde f mantenga la condición de cambio de signo. Es decir, si $I^{(0)} = (a, b)$ y $I^{(k)} = (a^{(k)}, b^{(k)})$ es el k-ésimo intervalo elegido el el paso k, entonces el intervalo que debe elegirse para el paso $k + 1$ es aquel que satisfaga la condición de cambio de signo en sus extremos del intervalo. Esta elección de los subintervalos hace que la secuencia de los puntos medios

$$x^{(k)} = \frac{a^{(k)} + b^{(k)}}{2},$$

se acerque asintóticamente a x_o, que es el valor que satisface $f(x_0) = 0$. En la figura 1 se muestran las primeras iteraciones como ejemplo.

a) Implemente un código de Fortran para encontrar los ceros de una función arbitraria, $f(x)$, en el intervalo $[a, b]$.

b) Determine la complejidad del algoritmo midiendo los pasos con las siguientes funciones

   1) $f(x) = x$ en $[-2, 1]$
   2) $f(x) = x$ en $[-20, 10]$
   3) $f(x) = \sin(x)$ en $(0, 2\pi)$

c) Con base en el inciso anterior, ¿es posible establecer una relación entre la precisión de su solución y el número de pasos?, ¿cuántos pasos debe hacer para disminuir un orden de magnitud en el error absoluto de aproximación?

Sugerencias: haga un módulo donde defina la función $f(x)$. Use un contador para los pasos.

9. Problema de Monty Hall (10 puntos)

El problema de Monty Hall es un problema de probabilidades. Se basa en un antiguo programa de televisión donde el presentador, Monty Hall, pedía al concursante elegir una de las tres puertas. El concursante se llevaba el premio, un automóvil o una cabra, que estaba oculto detrás de las puertas. Las reglas *modernas* del juego son las siguientes

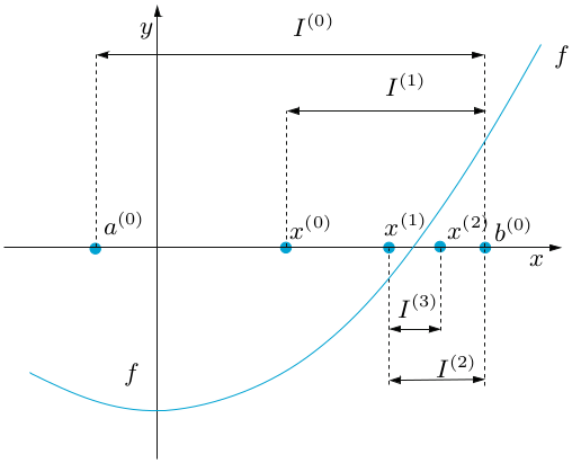

Figura 1: Primeras iteraciones del método de bisección. Fuente: Quarteroni et al. Scientific Computing with MATLAB and Octave.

1. Al concursante se le ofrece la posibilidad de elegir una de las tres puertas. Los premios son un automóvil (que está detrás de una de las puertas) o una cabra (hay una detrás de cada una de las otras dos puertas).

2. Después de que el concursante ha elegido una puerta, el presentador abrirá una de las otras dos puertas, siempre para mostrar una cabra.

3. Se ofrece al concursante la posibilidad de cambiar su elección inicial y escoger la otra puerta que descartara inicialmente y que continúa cerrada.

La pregunta es *¿se debe hacer el cambio o no?*. El problema tiene soluciones usando análisis probabilístico (probabilidad condicional) y también se puede resolver computacionalmente.

a) Escriba un diagrama de flujo del problema

b) Implemente un código de Fortran que le permita responder si se debe hacer el cambio o no. (Ayuda: necesita generar números aleatorios)

c) Si es posible usar programación en paralelo. ¿Qué esquema se debe usar?

d) Si va a usar *paralelismo de datos*. ¿Cómo va a distribuir las semillas aleatorias entre los distintos procesos paralelos?

Sugerencia: haga estadísticas primero suponiendo que va a hacer el cambio y después suponiendo que no lo va a hacer. Compare el número de veces que gana dado que hizo el cambio con el número de veces que gana dado que no hizo el cambio.

10. Números de Armstrong (10 puntos)

En la teoría de números los números de Armstrong, en honor a Michael F. Armstrong, en una base $b > 1$ es un número cuya suma de sus propios dígitos elevada a la k-ésima potencia es igual al mismo número si k es el número de dígitos del número n. Por ejemplo en base decimal, $b = 10$, se tienen los siguientes números de Armstrong

- $1 = 1^1$,
- $153 = 1^3 + 5^3 + 3^3$,
- $370 = 1^3 + 7^3 + 0^3$,
- $407 = 4^3 + 0^3 + 7^3$,
- $1634 = 1^4 + 6^4 + 3^4 + 4^4$,
- $8208 = 8^4 + 2^4 + 0^4 + 8^4$
- $9474 = 9^4 + 4^4 + 7^4 + 4^4$

Los demás números ya tienen cuatro dígitos o más y se conocen como la secuencia A005188 en la Enciclopedia en línea de secuencias de enteros. En una base distinta a la decimal, por ejemplo para $b = 3$ y $k = 3$, el número $122 = 1^3 + 2^3 + 2^3$ es un número de Armstrong.

A los números de Armstrong también se les conoce como números narcisistas debido a que es posible definir una función, llamada función narcisista, $F_b : \mathbb{N} \to \mathbb{N}$ para un número natural n, en la base $b > 1$

$$F_b(n) = \sum_{i=0}^{k-1} d_i^k,$$

donde $k = \lfloor \log_b n \rfloor + 1$ es el número de dígitos en el número de base b y

$$d_i = \frac{(n \mod b^{i+1}) - (n \mod b^i)}{b^i},$$

es el valor de cada dígito del número. Entonces, con la definición de F_b, un número es narcisista o número de Armstrong si $F_b(n) = n$, que se conoce como un punto fijo de la función. Se considera que todos los números $0 \leq n < b$ son números de Armstrong triviales y todos los otros son números de Armstrong no triviales.

1. Escriba un código de Fortran para encontrar los números de Armstrong para una base $b > 1$ cualquiera. Puede iniciar buscando entre los números de 1 dígito, luego de 2 dígitos, 3 dígitos, etc. ¿Cuál es la complejidad de su algoritmo?

2. Se conoce que las secuencias de los números de Armstrong para una base b dada son conjuntos finitos:

base b	Número de elementos de la secuencia	Secuencia
2	2	
3	6	
4	12	`https://oeis.org/A010343`
5	18	`https://oeis.org/A010346`
6	31	`https://oeis.org/A010348`
7	60	`https://oeis.org/A010350`
8	63	`https://oeis.org/A010351`
9	59	`https://oeis.org/A010353`
10	89	`https://oeis.org/A005188`

¿Cuánto tiempo le toma a su código encontrar la cantidad de elementos en una secuencia en función de b?.

3. ¿Es posible resolver el problema de forma paralela? (Escriba los argumentos de cómo dividir las tareas o los datos).

4. Si la respuesta anterior es afirmativa. Implemente una versión paralela del código.

5. Si hizo el inciso anterior, compare los tiempos, paralelo y secuencial, que le toma a su código encontrar la cantidad de elementos en una secuencia en función de b.

Sugerencia: haga estimaciones del tiempo que necesita su código, en el peor caso, para encontrar todos los números de Armstrong de k dígitos en base b se necesitan $O(b^k)$ pasos.