Unidad 0

Elementos de la computación numérica.

Ejercicio 1. (a) Mostrar que

$$(13.25)_{10} = (1101.01)_2 = (15.2)_8 = (\mathrm{D}.4)_{16}.$$

(b) Determinar la representación binaria de los siguientes números

a) 29, b) 0.625, c) 0.1, d) 5.75.

Ejercicio 2. Considere el conjunto de números de punto flotante $\mathbb{F}(2, 3, -1, 2)$.

(a) Determinar $x_{\text{mín}}$, $x_{\text{máx}}$, ϵ_M y el número de elementos de $\mathbb{F}$.

(b) Determinar los números de punto flotante positivos del conjunto $\mathbb{F}$.

(c) Determinar el conjunto de los números de punto flotante denormalizados positivos asociados a $\mathbb{F}$.

(d) Graficar sobre la recta real el conjunto de números de puntos flotantes determinados.

Ejercicio 3. Determinar los valores de $x_{\text{mín}}$, $x_{\text{máx}}$, ϵ_M para la representaciones de precisión simple y doble de la norma IEEE754 y mostrar que en la representación de precisión simple de la norma IEEE754 el número de dígitos *decimales* significativos es alrededor de 7, mientras que para la precisión doble es alrededor de 16.

Ejercicio 4. Determinar la representación de punto flotante de simple precisión de

a) -118.625, b) 0.1.

Ejercicio 5. Fortran 90 dispone de un conjunto de funciones intrínsecas para determinar las propiedades de la representación de punto flotante implementada en una computadora. Utilizando el siguiente programa verifique que en una computadora personal los datos de tipo numérico real en Fortran son efectivamente números de punto flotante de precisión simple de la norma IEEE754, mientras que los datos de tipo numérico de doble precisión son números de punto flotante de precisión doble según de dicha norma.

```
program machar
implicit none
integer i
real x
double precision w

write(*,*) 'Representacion datos numéricos'
write(*,*) ' base  = ', radix(i)

write(*,*) 'Datos de tipo real'
write(*,*) ' t     = ', digits(x)
write(*,*) ' L     = ', minexponent(x)
write(*,*) ' U     = ', maxexponent(x)
write(*,*) ' x_max = ', huge(x)
write(*,*) ' x_min = ', tiny(x)
write(*,*) ' eps_M = ', epsilon(x)

write(*,*) 'Datos de tipo doble precision'
write(*,*) ' t     = ', digits(w)
write(*,*) ' L     = ', minexponent(w)
write(*,*) ' U     = ', maxexponent(w)
write(*,*) ' x_max = ', huge(w)
write(*,*) ' x_min = ', tiny(w)
write(*,*) ' eps_M = ', epsilon(w)

end
```

Ejercicio 6. Explicar, a la luz de la representación de los números de punto flotante, los resultados obtenidos por los *bugs* cometidos en el siguiente programa.

```
program wrong
double precision d

d = 1.66661
write(*,*) d

call display(1)

end

subroutine display(x)
real x
write(*,*) x
end
```

Ejercicio 7. Considere la suma de cuatro números positivos:

$$y = x_1 + x_2 + x_3 + x_4.$$

(a) Mostrar que si los x_i son números de punto flotante decimales con una aritmética de t dígitos el error de redondeo en y está acotado por

$$|\Delta y| \leq (3x_1 + 3x_2 + 2x_3 + x_4)\, 5 \times 10^{-t}.$$

(b) Mostrar que el error de redondeo se minimiza reacomodando los números a sumar de manera que los más pequeños sean los que se sumen primero.

(c) Implementar un programa Fortran para evaluar la suma

$$\sum_{n=1}^{10\,000\,000} 1/n,$$

primero en el orden usual y luego en el orden opuesto. Considere primero los cálculos en simple precisión y luego en doble precisión. Explique a que se debe las diferencias obtenidas.

Ejercicio 8. Considere la adición de cuatro números positivos aproximadamente iguales

$$y = x_1 + x_2 + x_3 + x_4,$$

donde

$$x_i = x_0 + \delta_i, \quad i = 1, 2, 3, 4$$

siendo

$$|\delta_i| \ll x_0.$$

(a) Mostrar que si los x_i son números de punto flotante decimales con una aritmética de t dígitos el error de redondeo en y está acotado aproximadamente por

$$|\Delta y| \lesssim 4.5 \times 10^{1-t}\, x_0.$$

(b) Mostrar que si los números se suman como

$$y = (x_1 + x_2) + (x_3 + x_4),$$

efectuando primero las operaciones entre paréntesis, el error relativo en y está acotado aproximadamente por

$$|\Delta y| \lesssim 4 \times 10^{1-t}\, x_0.$$

(c) Como ejemplo numérico considere los números

$$x_1 = 0.5243 \times 10^0,$$
$$x_2 = 0.5262 \times 10^0,$$
$$x_3 = 0.5226 \times 10^0,$$
$$x_4 = 0.5278 \times 10^0.$$

Compare los resultados de $y = x_1 + x_2 + x_3 + x_4$ e $y = (x_1 + x_2) + (x_3 + x_4)$ al utilizar aritmética de cuatro dígitos.

Ejercicio 9. Supóngase que x e y son números positivos correctamente redondeados a t dígitos. Mostrar que la magnitud del error relativo de redondeo en $z = x - y$ está acotada por

$$\left|\frac{\Delta z}{z}\right| \leq \mathbf{u} + \frac{|x| + |y|}{|x - y|}\,(1 + \mathbf{u})\mathbf{u}.$$

Luego, si x e y son aproximadamente iguales, los errores de redondeo de x e y pueden propagarse de manera tal que el error relativo en z puede ser grande aunque el error absoluto sea pequeño (*fenómeno de cancelación de dígitos significativos*).

Ejercicio 10. El siguiente ejemplo muestra que la pérdida de precisión en la resta de dos números aproximadamente iguales puede tener un efecto drástico en expresiones que contengan dicha sustracción. La fórmula cuadrática nos dice que las raíces de $ax^2 + bx + c = 0$ son

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Si $b^2 \gg 4ac$ entonces, cuando $b > 0$ el cálculo de x_1 involucra en el numerador la sustracción de dos números casi iguales, mientras que si $b < 0$ esta situación ocurre para el cálculo de x_2. "Racionalizando el numerador" se obtienen las siguientes fórmulas alternativas que no sufren de este problema:

$$x_1 = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}},$$

siendo la primera adecuada cuando $b > 0$ y la segunda cuando $b < 0$.

Utilizar la fórmula usual y la "racionalizada" para calcular las raíces de

$$x^2 + 62.10x + 1 = 0,$$

utilizando aritmética decimal a cuatro dígitos. Interprete sus resultados según lo expuesto.

Ejercicio 11. Efectúe con un programa en Fortran los cálculos, matemáticamente equivalentes,

$$\text{a)} \sum_{i=1}^{1\,000\,000} 0.1, \quad \text{b)} \sum_{j=1}^{1\,000} \sum_{i=1}^{1\,000} 0.1.$$

Mostrar que en simple precisión la propagación del error de redondeo de la representación de 0.1 conduce a un error de redondeo acumulado en a) orden del 1 %, pero es mucho menor en b). Mostrar que el efecto de acumulación del error disminuye si se emplea doble precisión.

Ejercicio 12. Supóngase que se necesita generar una grilla en el intervalo $[a, b]$ consistente de $n + 1$ puntos igualmente espaciados con un paso $h = (b - a)/n$.

(a) ¿Cuál de los siguientes métodos es más apropiado en una aritmética de punto flotante?

$$x_0 = a, \quad x_k = x_{k-1} + h, \quad k = 1, \ldots, n$$

ó

$$x_k = a + kh, \quad k = 0, \ldots, n$$

(b) Implementar ambos métodos en un programa y dar un ejemplo que ilustre la diferencia entre ellos.

Ejercicio 13. (a) Mostrar que el error computacional (error de truncamiento + error de redondeo) de la fórmula de diferencia finita

$$\frac{f(x+h) - f(x)}{h},$$

para calcular $f'(x)$ está acotado por

$$\frac{2\delta}{h} + \frac{Mh}{2},$$

siendo δ una cota para el error (absoluto) de redondeo y M una cota para f''. Mostrar que, entonces, el valor óptimo de h, definido como aquel para el cual la cota del error computacional toma su valor mínimo, es $h_{\text{ópt}} = 2\sqrt{\delta/M}$.

(b) Implementar un programa para estimar la derivada de una función dada en un punto con la fórmula de diferencia finita del punto anterior. Testear el programa utilizando la función $\operatorname{sen}(x)$ para $x = 1$. Computar el error comparando con la función intrínseca $\cos(x)$. Graficar la magnitud del error como función de h, para $h = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ (utilizar escala logarítmica sobre ambos ejes) y determinar el valor de h que minimiza el error.

(c) Repetir el análisis utilizando ahora la fórmula de diferencia centrada

$$\frac{f(x+h) - f(x-h)}{2h},$$

para la cual el error computacional está acotado por

$$\frac{\delta}{h} + \frac{Mh^2}{6},$$

siendo $h_{\text{ópt}} = \sqrt[3]{3\delta/M}$.

Ejercicio 14. El polinomio $(x-1)^7$ tiene el valor cero en $x = 1$. Su forma expandida $x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$ es matemáticamente equivalente pero no lo es numéricamente. Computar y graficar los valores de este polinomio, utilizando ambas expresiones, en 241 puntos igualmente espaciados del intervalo $[0.988, 1.012]$, esto es, con un paso de 0.0001. Explicar el comportamiento observado.

Ejercicio 15. Implemente un programa para generar los primeros n términos de la sucesión dada por la ecuación de diferencias

$$x_{k+1} = 2.25\, x_k - 0.5\, x_{k-1},$$

con los valores iniciales

$$x_0 = \frac{1}{3}, \quad x_1 = \frac{1}{12}.$$

Tomar $n = 225$ en simple precisión. Confeccionar un gráfico semilogaritmico de los valores obtenidos como función de k. La solución exacta de la ecuación de diferencias es

$$x_k = \frac{1}{3}\left(\frac{1}{4}\right)^k.$$

la cual decrece monótonamente conforme k se incrementa. Analice y explique los resultados obtenidos (*Ayuda:* Encuentre la solución general de la ecuación de diferencias).

Ejercicio 16. El siguiente código cuenta el número de inversos multiplicativos imprecisos en la aritmética de punto flotante de la computadora.

```
program inversos
implicit none
integer NMAX
double precision ONE
parameter (NMAX=1000, ONE = 1.0d0)
double precision x,y,z
integer i,count

count = 0
do i=1,NMAX
   x = dble(i)
   y = ONE/x
   z = y*x
   if (z.ne.ONE) count = count + 1
end do
write(*,*) 'Incorrectos = ', count

end
```

Compilar el código con diferentes niveles de optimización utilizando tanto el compilador Fortran de GNU como el de Intel. Indicar las discrepancias obtenidas.

Ejercicio 17. El siguiente código tiene un *bug* para $x = 0$.

```
program bug
real signo
write(*,*) signo(0.0)
end

integer function signo(x)
real x
if (x.gt.0.0) then
   signo = 1
elseif (x.lt.0.0) then
   signo = -1
endif
end
```

Compilar el código sin y con optimización (de nivel 2). ¿Qué se observa?

Ejercicio 18. Dado el siguiente código

```
program loops
implicit none
integer n
parameter (n=5000)
double precision a(n,n)
integer i,j

do i = 1,n
   do j = 1,n
      a(i,j) = rand()
   enddo
enddo

end
```

utilizar el comando `time` para medir el tiempo de CPU requerido en su ejecución. Modificar el código intercambiando los bucles y medir nuevamente el tiempo de CPU. Explicar la diferencia de tiempos obtenida.

Ejercicio 19. Dado el siguiente código

```
program improved
integer NMAX, MMAX
parameter (NMAX=5000, MMAX=5000)
real a(NMAX,MMAX)
integer cond1, cond2

cond1 = 100
cond2 = 100
do i=1, NMAX
   do j=1,MMAX
      if(cond1.eq.cond2) then
         a(i,j) = real(i + j)
      else
         a(i,j) = 0.0
      endif
   enddo
enddo

end
```

compare el tiempo de CPU al compilar el programa con diversos niveles de optimización automática. ¿Qué mejoras manuales pueden hacerse al código? Compare el tiempo de CPU para el código así optimizado con los obtenidos por optimización automática.

Ejercicio 20. El siguiente programa calcula la traspuesta de una matriz en la forma simple y directa dada por la definición.

```
program trasponer_matriz
  implicit none
  integer, parameter :: N=2048
  integer, parameter :: M=2048
  double precision   :: a(N,M)
  double precision   :: b(M,N)
  integer            :: i,j
  double precision   :: t0,t1

  ! Tomar matriz de números
  ! al azar (entre 0 y 1)
  call random_number(a)

  call cpu_time(t0)

  ! Transponer en forma directa
  do i = 1, n
     do j = 1,m
        b(i,j) = a(j,i)
     enddo
  enddo

  call cpu_time(t1)
  write(*,*) t1-t0

end program trasponer_matriz
```

Por su parte, el siguiente programa implementa un algoritmo de *trasposición por bloques*. Comparar los tiempos de CPU respecto del programa directo utilizando diversos tamaños de bloque. Graficar el tiempo de CPU en función del tamaño del bloque. Determinar si existe una elección óptima para el tamaño del bloque.

```
program transpose_matrix
  implicit none
  integer, parameter :: N = 2048  ! dimensiones de la
  integer, parameter :: M = 2048  ! matriz y traspuesta
  double precision   :: a(N,M)    ! matriz a trasponer
  double precision   :: b(M,N)    ! matriz traspuesta
  integer            :: i,j       ! índice de matrices
  integer            :: ib,jb     ! índice de bloques
  integer            :: nb,mb     ! número de bloques
  integer            :: bsize     ! dimensión del bloque
  integer            :: ioff,joff ! offset i,j
  double precision   :: bswp      ! variable temporal
  double precision, allocatable :: buf(:,:) ! bloque
  double precision   :: t0,t1     ! tiempos de CPU
  character(32)      :: arg       ! línea de comandos

  ! Leer el tamaño del bloque de la línea de comando
  if ( command_argument_count() == 0 ) then
   write(0,*) "Error: Indicar el tamaño del bloque"
   stop
  endif
  call get_command_argument(1,arg)
  read(arg,*) bsize

  ! Chequear que el tamaño es divisor de N y M
  if( mod(N,bsize) /= 0 ) then
     write(0,*) "El tamaño del bloque debe ser divisor de ", N
     stop
  end if
  if( mod(M,bsize) /= 0 ) then
     write(0,*) "El tamaño del bloque debe ser divisor de ", M
     stop
  end if

  ! Tomar una matriz de números al azar (entre 0 y 1)
  call random_number(a)

  ! Número de bloques
  nb=N/bsize
  mb=M/bsize

  ! Asignar memoria para el bloque
  allocate(buf(bsize,bsize))

  call cpu_time(t0)

  ! Trasposición por bloques
  do ib = 1, nb
     ioff = (ib-1) * bsize
     do jb = 1, mb
        joff = (jb-1) * bsize
        do j = 1, bsize
           ! Leer un bloque de la matriz
           do i = 1, bsize
              buf(i,j) = a(i+ioff, j+joff)
           enddo
        enddo
        ! Trasponer el bloque
```

```
         do j = 1, bsize
            do i = 1, j-1
               bswp     = buf(i,j)
               buf(i,j) = buf(j,i)
               buf(j,i) = bswp
            enddo
         enddo
         ! Almacenar el bloque transpuesto en
         ! la matriz transpuesta final
         do i=1,bsize
            do j=1,bsize
               b(j+joff, i+ioff) = buf(j,i)
            enddo
         enddo
      enddo
   enddo

   call cpu_time(t1)
   write(*,*) bsize,t1-t0

end program transpose_matrix
```