Informática II

Clase 3: Ecuaciones no lineales: Método de la Secante
Extras: Derivadas numéricas

Mario Merino Martínez
mario.merino@upm.es

Escuela de Ingeniería Aeronáutica y del Espacio
Universidad Politécnica de Madrid

5 de marzo de 2013

Índice

Diferencias finitas

Cuando sólo conocemos cuánto vale f en una serie de puntos, podemos **aproximar la derivada** f' a partir de ellos. Por ejemplo, sean $\{x_i\}$ puntos equiespacidos (con $h = x_i - x_{i-1}$ constante), y $\{f_i = f(x_i)\}$. Podemos aproximar f como si estuviese compuesta por pequeños segmentos (interpolación lineal). La **derivada en cada punto** puede aproximarse así:

$$f'(x_n) \simeq \frac{f_{n+1} - f_n}{h}; \quad f'(x_n) \simeq \frac{f_n - f_{n-1}}{h};$$

$$\text{o bien: } f'(x_n) \simeq \frac{f_{n+1} - f_{n-1}}{2h}$$

(diferencias adelantadas, atrasadas, y centradas, respectivamente).

Diferencias finitas II

De todas ellas, la centrada es mejor: el **error de truncación es** $\sim h^2$ **en lugar de** $\sim h$**.**
Análogamente, podemos aproximar la **segunda derivada** a partir de **tres puntos**, interpolando con una parábola:

$$f''(x_n) \simeq \frac{f_{n+1} - 2f_n + f_{n-1}}{h^2}$$

(también diferencias centradas y error $\sim h^2$)

Comentarios sobre derivación numérica I

- Existen **dos tipos de error**: error de truncación (por haber "aproximado" la formula de la derivada), y error de redondeo o de máquina (debido al número limitado de cifras significativas)
- El cálculo de derivadas de forma numérica e**stá mál condicionado, y ha de evitarse siempre que sea posible**. *Nunca* lo hacemos si conocemos f y podemos derivar "a mano".

Comentarios sobre derivación numérica II

- Esto es debido a que **estamos restando números muy próximos**, $f_{n+1} - f_n$ por ejemplo, lo cual da lugar a una pérdida de cifras significativas (gran error de redondeo):

$$119519235182.43\ldots - 119519235182.18\ldots =$$

$$= 0.25000000000000$$

(14 cifras sign. - 14 csifras dan solo 2 cifras correctas!)

- Para $f = O(1)$, cuando $h \lesssim 10^{-5}$ el error de redondeo ya supera al de truncación, y la solución empeora si seguimos reduciendo h.

Para más info: p. 119 de "Elementos de Cálculo Numérico", por D. Rivas y C. Vázquez.

Recordatorio sobre Arrays en Fortran

¿Cómo referirnos a los distintos elementos de un Array?

En la declaración:

- `real*8 :: A(4,5)` indica al compilador las dimensiones totales de la matriz A

En cualquier otro sitio:

- `A(4,5)` indica solamente el elemento 4,5.
- `A` indica toda la matriz; equivale a `A(:,:)`, `A(1:4,1:5)`
- `A(2,:)` indica toda la fila 2
- `A(:,4)` indica toda la columna 4
- `A(2:3,1:4)` indica la submatriz de las filas 2,3 y las columnas 1,2,3,4

`A(1,:) = (/ 1d0, 2d0, -7.5d0, 0d0, -1d2 /)` da valores a toda la fila 1 ("constructor de vectores")

Pasar funciones y subrutinas como argumentos

Cuando creamos una subrutina o una funcion, podemos no solo pasar como **argumentos** variables normales (reales, enteros...), sino **también otras subrutinas y funciones**:, declarándolos con el **atributo external**:

```
subroutine newton(x,fun,dfun,nmax,tol)
implicit none
real*8 :: x, tol
real*8, external :: fun, dfun
integer :: nmax
...
```

Lo mismo en el **programa principal**:

```
program main
implicit none
real*8, external :: fun1, dfun1
...
call newton(x,fun1,dfun1,...)
...
```

¿Argumentos de entrada o de salida? I

Al escribir una subrutina o función, por ejemplo LUsolve(A,b,n), los a**rgumentos pueden ser "de entrada", "de salida", o de "entrada-salida"**

- **A y n** son de entrada, ya que al llamar a LUsolve, le pasamos A factorizada y la dimension n, y no se modifican dentro de la subrutina.
- **b** es de entrada-salida, ya que al llamar a LUsolve pasamos el vector B y en esa misma variable recogemos el vector X.

¡Esto es confuso y peligroso! ¿Cómo podemos estar seguros que una subrutina o funcion "blubli(A,B,C,D,F)" no modifica e.g. el argumento D? Los **efectos** serían **desastrosos en el programa principal** si no contábamos con ello.

¿Argumentos de entrada o de salida? II

El **atributo intent** permite **especificar de qué tipo son los argumentos** de una funcion o subrutina. En el ejemplo anterior, tendríamos:

```
subroutine LUsolve(A,b,n)
real*8, intent(in) :: A(n,n)
real*8, intent(inout) :: b(n)
integer, intent(in) :: n
```

Con esto, conseguimos que:

- **El compilador se queje** si en la subrutina intentamos escribir sobre una variable intent in , o si olvidamos asignarle un valor a una variable intent out.
- Que **de un vistazo a la cabecera** de la subrutina o funcion, **sepamos** que e.g. A no será sobreescrita por LUsolve.

Método de la Secante: Justificación

- El método de Newton para resolver ecuaciones $f(x) = 0$ es muy eficiente (convergencia cuadrática), pero **requiere conocer explicitamente la derivada** de la función, $f'(x)$ y **evaluarla en cada paso**

- A veces es **muy costoso** (o no es posible) obtener f'

El **método de la secante** está basado en la misma idea que el método de Newton: *aproximar la función por una recta, e iterar*. Pero aproxima la derivada f' numéricamente con los dos últimos valores de x obtenidos:

$$f'(x) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Método de la Secante I

La **formula de iteración** del método de la **secante** es, por tanto:

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Esto es, x_{n+1} es el valor de x para el cual la recta secante que pasa por los puntos $[x_{n-1}, f(x_{n-1})]$ y $[x_n, f(x_n)]$ corta el eje. **Diferencias** fundamentales con el método de Newton:

- La velocidad de convergencia es menor que cuadrática: $\Delta x_n \sim (\Delta x_{n-1})^p$, con $p = \frac{1+\sqrt{5}}{2}$.
- **Se necesitan dos condiciones iniciales**, x_0 y x_1, para poder arrancar (en lugar de una)

Implementación del método

La **principal dificultad** de implementación: hay que arrastrar los dos últimos valores de x calculados y a**ctualizarlos** con cuidado:

```
do i=1,nmax
   ...
   x = ... ! Cálculo de la nueva x a partir de x_1 y x_2
   x_2 = x_1 ! Actualización de x_2 con x_1 para la próxima iteración
   x_1 = x ! Actualización de x_1 con x para la próxima iteración
   ...
end do
```

Como **criterio de parada** utilizaremos el mismo que en Newton:

$$\frac{|\alpha - x_i|}{|\alpha|} \simeq \frac{|x_{i+1} - x_i|}{|x_{i+1}|} < \varepsilon \quad y \quad |f(x_{i+1})| < \varepsilon$$

Ejemplo de Implementación

```
program secante
integer :: i, nmax=10
real*8 :: x, x_1, x_2, f, f_1, f_2, dx, p, eps = 1d-8
real*8, external :: fun
x_1 = 1d0 ! Aproximaciones iniciales
x_2 = 0.5d0
f_1 = fun(x_1)
f_2 = fun(x_2)
do i=1,nmax
  p = (f_1-f_2)/(x_1-x_2)! Pendiente
  dx = -f_1/p ! La corrección de x
  x = x_1 + dx ! Nuevo valor de x
  f = fun(x)! Nuevo valor de f(x)
  if (abs(dx/x)<eps .and. abs(f)<eps) then ! Criterio parada
    print*, 'solucion: ', x, ' en iteracion: ', i;     exit
! Sale del bucle
  end if
  x_2 = x_1 ! Actualizacion para la siguiente iteracion
  x_1 = x
  f_2 = f_1
  f_1 = f
end do
if (i>nmax) print*, 'error: no convergencia'
end program secante
```