Capítulo 1

Funciones

En Fortran hay funciones preprogramadas (intrínsecas) que ya las hemos visto: raíz cuadrada, funciones trigonométricas, etc. Pero además de estas funciones se le permite al programador crear las suyas. Existen dos métodos para construir funciones en Fortran: a través de la Función Sentencia o de la Función Externa.
Es muy útil poder crear funciones, ahorra tiempo, hace los programas más compactos, enfocados y elegantes. Además permite que uno utilice funciones que ya se encuentran disponibles en libros o en páginas de internet. Sobre esto último, existen sitios web donde se discute cuál es el mejor algoritmo para una tarea determinada y cuales son sus mejores implementaciones en distintos lenguajes. Las funciones han tomado históricamente distintos nombres (procedimientos, subprogramas, etc) en distintos lenguajes de programación pero siempre existe una manera de definirlas.

1.1. Función Sentencia

Consiste en una sola sentencia aritmética definida en el programa. Esta estructura existe en otros lenguajes, por ejemplo, en Python se la llama función Lambda.
Se las define al comienzo del programa dándole un nombre a la función y se las construye usando variables ficticias, es decir que no son las variables del programa, sólo se las usa como referencia para construir la fórmula matemática de la función y sólo valen en la sentencia. Es decir, para indicar el orden y en que lugar se encuentran las distintas variables de la fórmula a calcular.
La forma sería la siguiente:

función(var1,var,2,var3...) = expresión matemática usando las variables var1, var2, var3,...

Donde *función* es el nombre que se le da a la función, las *var1,var2, var3, etc* son las variables que se usan en la *expresión matemática*. Estas variables indican el orden de los argumentos con los que defino la función. Por eso su orden es muy importante en el momento de llamarla.

Ejemplos:
Si se necesita una función discriminante (para ecuaciones cuadráticas), la puedo construir de esta manera al comienzo del programa (donde defino los tipos de variables del programa) escribiendo:

```
DISC(A,B,C) = B**2-4.*A*C
```

Entonces en el programa se puede escribir:

```
Program prueba_de_funcion_sentencia
```

```
DISC(A,B,C) = B**2-4.*A*C
⋮
X= DISC(24.5, 34.5, 67.8) +25.4*C3+MAG
⋮
MAGROJA = SQRT(DISC(z1+3, 45.5*8, (z2+z4+f6)+28)
⋮
```

Detalles a tener en cuenta:

- Las variables (var1,var2,...) pueden ser de cualquier tipo incluyendo variables lógicas o de caracteres.
- Las funciones para el programa son variables, es decir la función es una variable que cuando se llama realiza un cálculo y entrega el resultado de su ejecución, en vez de ir a buscar un número que este guardado en la memoria. Por lo tanto una función, ya que es una variable, puede ser integer, real*4, real*8, complex, character o logical, etc.

1.1.1. Ejemplo de aplicación de Funciones - Integral por trapecios

De los cursos de análisis matemático uno adquiere la idea de que la única manera de resolver los problemas pasa por obtener siempre su solución analítica. Pero en la vida real, esa situación es más bien rara, o imposible. Incluyendo el hecho que en el caso particular de una integral puede que esta no tenga primitiva o que sea muy complejo o laborioso encontrarla. Más aún si lo que se lo quiere resolver es una ecuación diferencial. Por esta razón, en problemas muy complejos se usan los sistemas de computación con el fin aproximar la solución, es decir obtener una solución numérica. Esta puede ser en algunos casos una solución exactas o a veces aproximada. Desarrollaremos un caso en particular como ejemplo.

Uno de los métodos más simples de aproximar numéricamente el resultado de una integral definida es el método de los trapecios. Vamos a ver con un ejemplo cómo funciona este método desde el punto de vista computacional. Un tratamiento más profundo de sus propiedades incluyendo ventajas y desventajas se estudia en cursos de análisis numérico.

Con este método lo que se desea es integrar la función f(x) en el intervalo [a, b], es decir queremos medir el área bajo la curva. Para ello dividiremos el intervalo en varios subintervalos más pequeños y veremos cómo se podría aproximar con un trapecio al área bajo la curva de la función f(x). Ya que la idea del método es dividir el área en n subintervalos de tamaño h, por lo que h = (b − a)/n. Si llamo x_i a los puntos que separan cada uno de estos subintervalos tendríamos que:
a = x_0, x_i = x_0 + ih y b = x_n
En la figura 1.1 veamos cómo sería esta situación con un dibujo de la función f(x) entre los límites x_i y x_{i+1}, (donde x_{i+1} = x_i + h).
:

Entonces, el área para el trapecio del la figura es:

$$A_i = hf(x_i) + \frac{h}{2}[f(x_{i+1}) - f(x_i)] = \frac{h}{2}[f(x_{i+1}) + f(x_i)]$$

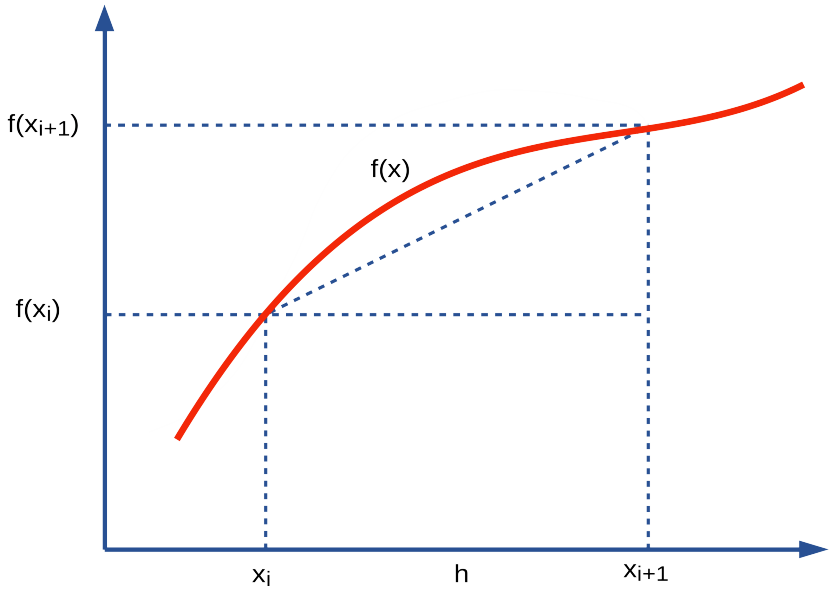

Figura 1.1. Aproximación por trapecios

y la aproximación a la integral será la suma de todos los trapecios A_i

$$Integral = \sum_{i=0}^{n} A_i$$

si hago todos lo reemplazos, obtengo:[1]

$$Integral = \frac{h}{2}[f(a) + f(b)] + h\sum_{i=1}^{n-1} f(x_i)$$

Veamos el programa que haría este cálculo y tomemos con función $f(x) = x^2$
y la integraremos en el intervalo [0,1]
En forma teórica:

$$\int_0^1 x^2 dx = 1/3$$

Veamos como sería un programa y la precisión de los resultados.

```
program x2
real*8 integral
f(z)=z*z
```

[1] Regla mnemotécnica: La suma extremos dividido dos más la suma de los puntos intermedios y todo esto multiplicado por el intervalo

```
x0=0.
xn=1.
read(*,*)n
h=(xn-x0)/n
integral=0.

do i=1,n-1
   x=i*h
   integral=integral+f(x)
enddo

integral=integral*h+(f(xn)+f(x0))*h/2
write(*,*)'Para ', n,' intervalos, la integral es=',integral
end
```

Si corro este programa para distintos valores de la cantidad de intervalos, obtengo los resultados del tabla 1.1. Como puede verse en la tabla no sirve pensar que se puede llegar al límite infinito sumando una cantidad gigantesca de trapecios debido a los errores de redondeo. Un buen resultado se obtiene cerca del 1,000,000 de trapecios, tomando intervalos más pequeños el resultado se deteriora. Pero por otro lado, la tabla nos muestra que según lo que se requiera de precisión en el resultado hay formas de evaluar este problema, y además de encontrar y determinar un resultado aproximado muy bueno incluyendo alguna idea de la precisión. En este caso hemos obtenido la integral con un error 10^{-7}.
En este programa usamos f(x) = x^2 como función a integrar. Si quisiera integrar otra función en otro rango de valores, sólo tendría que cambiar la definición de la función por otra y modificar los límites del intervalo a integrar. El programa no requiere ninguna otra modificación. Sólo hay quecambiar la función y volver a compilarlo.

Cantidad de intervalos	Resultado de la aproximación de la integral
10	0.33500001696869747
100	0.33334997741140865
1,000	0.33333354714617608
10,000	0.33333330969899705
100,000	0.33333330800677946
1,000,000	0.33333333080419814
10,000,000	0.33333334502431866
100,000,000	0.33333332725965703
1,000,000,000	0.33333330505514902

Tabla 1.1. Resultados de sumar N cantidad de trapecios. Puede notarse que a partir de cierto límite en el tamaño de los intervalos la aproximación deja de funcionar.

El método de trapecios, no es el método más usado pero si es un método que tiene interés académico porque sirve para entender la complejidad del cálculo y determinar cotas teóricas al problema de aproximar la integral. Existen métodos muchos mejores que el de trapecios y modificaciones para hacerlo mucho más eficiente.

1.1.2. Función Externa

La función sentencia que vimos en la sección anterior es muy útil pero tiene sus limitaciones. La mayor de ellas es que sólo permite una fórmula matemática que se escriba en un solo renglón. Por ejemplo, una función a trozos no se podría programar como función sentencia, ya que se necesitarían al menos más de un renglón para programarla con el comando IF().
En cambio, la función externa, trabaja como si fuera otro programa y este es llamado por el programa principal al igual que la función sentencia. Un esquema simple, sería:

```
program Principal
real*4 F
⋮
A= F(x)+5.0
⋮
END
Function F(x)
⋮
F = ···
RETURN
END
```

Como se ve en el esquema, la función fue escrita en el mismo archivo del programa, después de la sentencia END[2]. Notar que en general la función tiene la misma estructura que un programa salvo por el inicio como **FUNCTION nombre(variables que recibe)** y la orden **RETURN** que devuelve la ejecución hacia el programa principal.

Detalles que definen a la función externa:

- Puede haber muchas funciones que son utilizadas por un programa.
- Las funciones tienen variables internas que guardan datos que son de la función, cualquier modificación que se realice sobre ellas no afecta las variables del programa principal aunque tengan el mismo nombre), con la excepción del nombre de la función que retorna con el valor del resultado al programa principal).
- La orden **RETURN** devuelve la ejecución al lugar exacto donde la función fue llamada. Un programa puede llamar repetidamente a la función desde distintos lugares.
- Las funciones externas, al igual que la función sentencia son para el programa principal variables y cumplen con las reglas de estas. Según la primera letra de su nombre serán enteras o reales y se deberá definir cualquier otro tipo en forma explícita. Ejemplo: Si escribo Complex G(x,y) en el programa principal, en el comienzo de la función se deberá también escribir Complex Function G(x,y), para definirla como compleja.
- La función cuando es llamada depende para realizar su cálculo de las variables de referencia que se encuentran dentro del paréntesis. A estas variables se las denomina argumentos de la función. Los nombres de estos argumentos en el programa que llama a la función, pueden o no coincidir con los que se usan en la función para hacer los cálculos. Es decir, pueden

[2]Puede escribirse en un archivo aparte, pero debe compilarse con el programa que la utiliza. Ejemplo: gfortran programa.f funcion1.f funcion2.f -o programa.

tener otros nombres, ya que en realidad se copian en el momento de la llamada a la función. Pero tienen que ser del mismo tipo de variable. Si el primer argumento es un entero, también debe ser un entero en la función, lo mismo para el segundo o tercero y así. Incluso pueden ser variables dimensionadas y en ese caso no sólo debe coincidir el tipo de variable sino la dimensión y debe repetirse la definición de dimensión en la función. Ejemplo:

```
programa principal
REAL*4 A(100)
Complex C
⋮
Z = F(A,C)
⋮
END
Function F(X2,B)
REAL*4 X2(100)
COMPLEX B
⋮
RETURN
END
```

En este ejemplo, la función fue escrita de argumentos X2 y B, pero en el programa principal se la llama con las variables A y C. Donde A es vector de 100 elementos y C es un número complejo. Por eso X2 y B son definidas en la función del mismo tipo que las variables del programa principal. Note que el orden es quien decide cuál variable se copia a la correcta, en este caso la primera es el vector y la segunda es el complejo. El orden en el llamado determina cuál variable es cuál en la función.

- Las funciones fueron inventadas para el Fortran y copiadas con modificaciones en todos los lenguajes más modernos. Se las suele llamar en distintas encarnaciones como procedimientos, subprogramas, etc. En el caso de Python, que veremos más adelante en la cursada, se viola la idea de función matemática ya que estas pueden devolver mas de un valor. Pero en sí, son muy similares a las de Fortran.

Por ejemplo, si programamos una función que nos calcule la serie $\sum_{i=1}^{n} 1/i^k$, donde básicamente le enviamos N, K y nos devuelve la suma de la serie, sería así:

```
Function Serie(n,k)

Serie=0.
DO i=1, n
  x = i
  Serie = Serie + 1/ x**k
ENDDO

RETURN
```

end