6.6.1. Integración numérica

Vamos a implementar un programa de integración numérica que aproxime el valor de

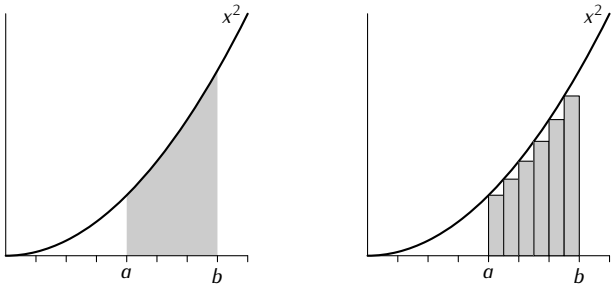
$$\int_a^b x^2\,dx$$

con la fórmula

$$\sum_{i=0}^{n-1} \Delta x \cdot (a + i \cdot \Delta x)^2,$$

donde $\Delta x = (b - a)/n$. El valor de n lo proporcionamos nosotros: a mayor valor de n, mayor precisión en la aproximación. Este método de aproximación de integrales se basa en el cálculo del área de una serie de rectángulos.

En la gráfica de la izquierda de la figura que aparece a continuación se marca en gris la región cuya área corresponde al valor de la integral de x^2 entre a y b. En la gráfica de la derecha se muestra en gris el área de cada uno de los 6 rectángulos ($n = 6$) utilizados en la aproximación. La suma de las 6 áreas es el resultado de nuestra aproximación. Si en lugar de 6 rectángulos usásemos 100, el valor calculado sería más aproximado al real.



La función Python que vamos a definir se denominará *integral_x2* y necesita tres datos de entrada: el extremo izquierdo del intervalo (*a*), el extremo derecho (*b*) y el número de rectángulos con los que se efectúa la aproximación (*n*).

La cabecera de la definición de la función será, pues, de la siguiente forma:

integral.py

```
1 def integral_x2(a, b, n):
2     ...
```

¿Qué ponemos en el cuerpo? Pensemos. En el fondo, lo que se nos pide no es más que el cálculo de un sumatorio. Los elementos que participan en el sumatorio son un tanto complicados, pero esta complicación no afecta a la forma general de cálculo de un sumatorio. Los sumatorios se calculan siguiendo un patrón que ya hemos visto con anterioridad:

integral.py

```
1 def integral_x2(a, b, n):
2     sumatorio = 0
3     for i in range(n):
4         sumatorio += lo que sea
```

Ese «*lo que sea*» es, precisamente, la fórmula que aparece en el sumatorio. En nuestro caso, ese fragmento del cuerpo de la función será así:

integral.py

```
1 def integral_x2(a, b, n):
2     sumatorio = 0
3     for i in range(n):
4         sumatorio += deltax * (a + i * deltax) ** 2
```

Mmmmm... En el bucle hacemos uso de una variable *deltax* que, suponemos, tiene el valor de Δx. Así pues, habrá que calcular previamente su valor:

integral.py

```
1 def integral_x2(a, b, n):
2     deltax = (b-a) / n
3     sumatorio = 0
4     for i in range(n):
5         sumatorio += deltax * (a + i * deltax) ** 2
```

La variable *deltax* (al igual que *i* y *sumatorio*) es una variable local.

Ya casi está. Faltará añadir una línea: la que devuelve el resultado.

integral_5.py — integral.py

```
1 def integral_x2(a, b, n):
2     deltax = (b-a) / n
3     sumatorio = 0
4     for i in range(n):
5         sumatorio += deltax * (a + i * deltax) ** 2
6     return sumatorio
```

¿Hecho? Repasemos, a ver si todo está bien. Fíjate en la línea 2. Esa expresión puede dar problemas:

1. ¿Qué pasa si *n* vale 0?
2. ¿Qué pasa si tanto *a*, como *b* y *n* son enteros?

Vamos por partes. En primer lugar, evitemos la división por cero. Si *n* vale cero, el resultado de la integral será 0.

integral_6.py — integral.py

```
1 def integral_x2(a, b, n):
2     if n == 0:
3         sumatorio = 0
4     else:
5         deltax = (b-a) / n
6         sumatorio = 0
7         for i in range(n):
8             sumatorio += deltax * (a + i * deltax) ** 2
9     return sumatorio
```

Y, ahora, nos aseguraremos de que la división siempre proporcione un valor flotante, aun cuando las tres variables, *a*, *b* y *n*, tengan valores de tipo entero:

integral_7.py — integral.py

```
1 def integral_x2(a, b, n):
2     if n == 0:
3         sumatorio = 0
4     else:
5         deltax = (b-a) / float(n)
6         sumatorio = 0
7         for i in range(n):
8             sumatorio += deltax * (a + i * deltax) ** 2
9     return sumatorio
```

Ya podemos utilizar nuestra función:

integral.py — integral.py

```
    ⋮
11 inicio = float(raw_input('Inicio del intervalo: '))
12 final = float(raw_input('Final del intervalo: '))
13 partes = int(raw_input('Número de rectángulos: '))
14
15 print 'La integral de x**2 entre %f y %f' % (inicio, final),
16 print 'vale aproximadamente %f' % integral_x2(inicio, final, partes)
```

En la línea 16 llamamos a *integral_x2* con los argumentos *inicio*, *final* y *partes*, variables cuyo valor nos suministra el usuario en las líneas 11–13. Recuerda que cuando llamamos a una función, Python asigna a cada parámetro el valor de un argumento siguiendo el orden de izquierda a derecha. Así, el parámetro *a* recibe el valor que contiene el argumento *inicio*, el parámetro *b* recibe el valor que contiene el argumento *final* y el parámetro *n* recibe el valor que contiene el argumento *partes*. No importa cómo se llama cada argumento. Una vez se han hecho esas asignaciones, empieza la ejecución de la función.

Un método de integración genérico

El método de integración que hemos implementado presenta un inconveniente: sólo puede usarse para calcular la integral definida de una sola función: $f(x) = x^2$. Si queremos integrar, por ejemplo, $g(x) = x^3$, tendremos que codificar otra vez el método y cambiar una línea. ¿Y por una sóla línea hemos de volver a escribir otras ocho?

Analiza este programa:

integracion_generica.py

```
1  def cuadrado(x):
2      return x**2
3
4  def cubo(x):
5      return x**3
6
7  def integral_definida(f, a, b, n):
8      if n == 0:
9          sumatorio = 0
10     else:
11         deltax = (b-a) / float(n)
12         sumatorio = 0
13         for i in range(n):
14             sumatorio += deltax * f(a + i * deltax)
15     return sumatorio
16
17 a = 1
18 b = 2
19 print 'Integración entre %f y %f' % (a, b)
20 print 'Integral de x**2:', integral_definida(cuadrado, a, b, 100)
21 print 'Integral de x**3:', integral_definida(cubo, a, b, 100)
```

¡Podemos pasar funciones como argumentos! En la línea 20 calculamos la integral de x^2 entre 1 y 2 (con 100 rectángulos) y en la línea 21, la de x^3. Hemos codificado una sola vez el método de integración y es, en cierto sentido, «genérico»: puede integrar cualquier función.

Pon atención a este detalle: cuando pasamos la función como parámetro, no usamos paréntesis con argumentos; sólo pasamos el nombre de la función. El nombre de la función es una variable. ¿Y qué contiene? Contiene una referencia a una zona de memoria en la que se encuentran las instrucciones que hemos de ejecutar al llamar a la función. Leer ahora el cuadro «Los paréntesis son necesarios» (página 245) puede ayudarte a entender esta afirmación.