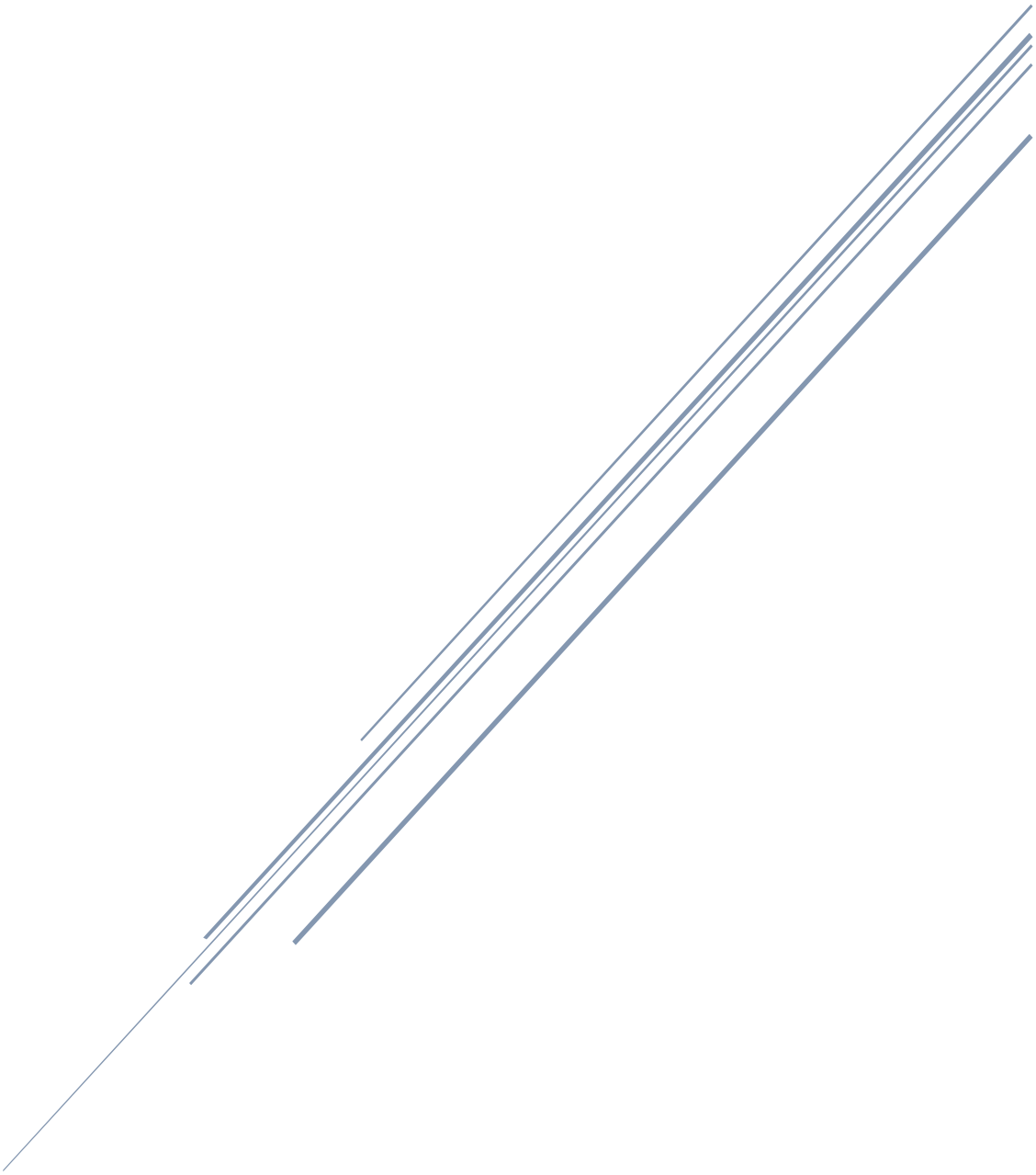
APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 0



FDI-UCM
Iván Aguilera Calle – Daniel García Moreno

1. Objetivo

El objetivo de esta primera práctica es realizar una primera toma de contacto con Octave realizando para ello la implementación de un algoritmo de integración numérica basado en el método de Monte Carlo.

$$I \approx \frac{N_{Debajo}}{N_{Total}} \times (b-a) \times M$$

M: es el máximo de la función en el intervalo [a, b].

NDebajo: es el número de puntos (x,y) generados aleatoriamente cuya coordenada y es menor que el valor de la función f(x) para ese valor de x.

NTotal: es el número total de puntos generados aleatoriamente dentro del rectángulo.

2. Desarrollo e implementación

```
function I = mcint(fun, a, b, num_puntos)
  maximo = dameMaximo(fun, a, 0.01, b);
  puntosRandom = generaPuntosAleatorios(a, b, maximo, num_puntos);
  plot(puntosRandom(1,:),puntosRandom(2,:))
  tic();
  nDebajo = numPuntosDebajoIterativo(fun, puntosRandom);
  I = (nDebajo/num_puntos) * (b - a) * maximo
  tiempo_iterativo = toc()

  tic();
  nDebajo = numPuntosDebajoVectores(fun, puntosRandom);
  I = (nDebajo/num_puntos) * (b - a) * maximo
  tiempo_vectores = toc()

endfunction
```

Figura 1

La ejecución comienza ejecutando la función mcint, que nos da un valor aproximado de la integral de la función “fun” en el intervalo “a, b” generando para ello “num_puntos” aleatorios en el intervalo [a, b]. Para ello se realizan una serie de operaciones que se observan en la Figura 1. Lo primero consiste en obtener el máximo de la función a integrar en el intervalo. Para ello se llama a la función dameMaximo, descrita en el Figura 2.

```
function maximo = dameMaximo(f, inf, d, sup)
  valores = inf:d:sup;
  maximo = max(f(valores));
endfunction
```

Figura 2

La función dameMaximo recibe la función “f”, valor inferior y superior del intervalo a tráves de “inf” y “sup” respectivamente. Para el cáculo del máximo se van a generar una serie de puntos con una separación de “d”. Entre los puntos generados nos quedaremos con el que tenga un mayor valor.

```
function puntos = generaPuntosAleatorios(inf, sup, maximo, numPuntos)
  puntos = [inf + (sup - inf) * rand(1, numPuntos); 0 + (maximo - 0) *
rand(1, numPuntos)];

  #La primera fila de la matriz son las coordenadas X
  #y la segunda fila las coordenadas Y de los puntos
endfunction
```

Figura 3

A continuación generamos una serie de puntos aleatorios tal que inf <= x <= sup y 0 <= y <= máximo y lo guardamos en una matriz de 2 filas tal que la primera fila contiene las coordenadas X de los puntos y la segunda fila contiene las coordenadas Y, tal como se observa en la Figura 3.

```
function num = numPuntosDebajoIterativo(f, puntosRandom)
  num = 0;

  for i = 1:columns(puntosRandom)
    fy = f(puntosRandom(1, i));
    y = puntosRandom(2, i);
    if (fy > y)
      num++;
    endif
  endfor

endfunction
```

Figura 4

Una vez generada la matriz de puntos aleatorios “puntosRandom”, vamos a ver cuántos de estos puntos se encuentran por debajo de la función “f”. La implementación se ha realizado de maneras distintas.

Como se aprecia en la Figura 4, se ha seguido un algoritmo iterativo que realiza num_puntos iteraciones a partir de un bucle donde se va a consultar coordenada a coordenada los puntos y comprobar que están por debajo de la función, llevando para ello un contador. Es entonces que “fy” almacena la imagen que tiene un punto generado aleatoriamente dentro de la función. Si el valor de “fy” es mayor que la coordenada “y” de un punto, es que estamos ante un punto que se encuentra por debajo de la función.

```
function num = numPuntosDebajoVectores(f, puntosRandom)
  fy = f(puntosRandom(1, :));
  y = puntosRandom(2, :);

  num = length(find(fy > y))
endfunction
```

Figura 5

En la Figura 5, el problema se ha tratado mediante operaciones con vectores.

3. Prueba del algoritmo

Tras haber implementado las partes necesarias pasamos a realizar la prueba de ejecución indicada en la Figura 6.

```
#PRUEBA 1 - f(x) = x^2
mcint(@(x)(x.^2),0,3,10000)
quad(@(x)(x.^2), 0 ,3)
```

Figura 6

La prueba que hemos realizado consiste en el cálculo de la integral de una función parabólica. Tras la ejecución podemos observar en la Figura 7, que el tiempo utilizado en el cálculo del número de puntos que se encuentran por debajo del función para calcular el valor estimado de la integral es superior en la implementación iterativa que en la que se utiliza operaciones con vectores.

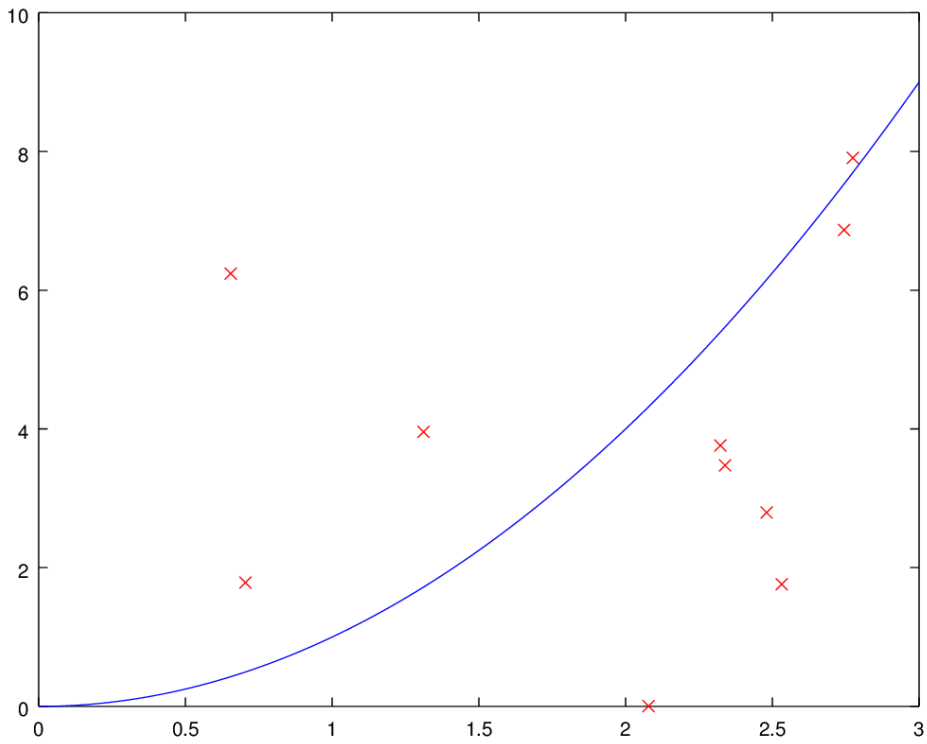
```
>> mcint(@(x)x.^2, 0, 3, 10000)
I =  9.1503
tiempo_iterativo =  0.21715
num =  3389
I =  9.1503
tiempo_vectores =  6.5589e-04
ans =  9.1503
```

Figura 7

Para comprobar que la integral obtenida por mcint es correcto, ejecutamos la función quad para la misma función parabólica y vemos que el valor es bastante aproximado. Si incrementamos el número de puntos generados aleatoriamente también incrementaremos la precisión de la función mcint.

```
>> quad(@(x)x.^2, 0, 3)
ans =  9.0000
```

Figura 8



Función x^2 y 10 puntos random