montecarlo_iterativo

June 19, 2019

In [12]:
```
#José Antonio Garrido Sualdea
```

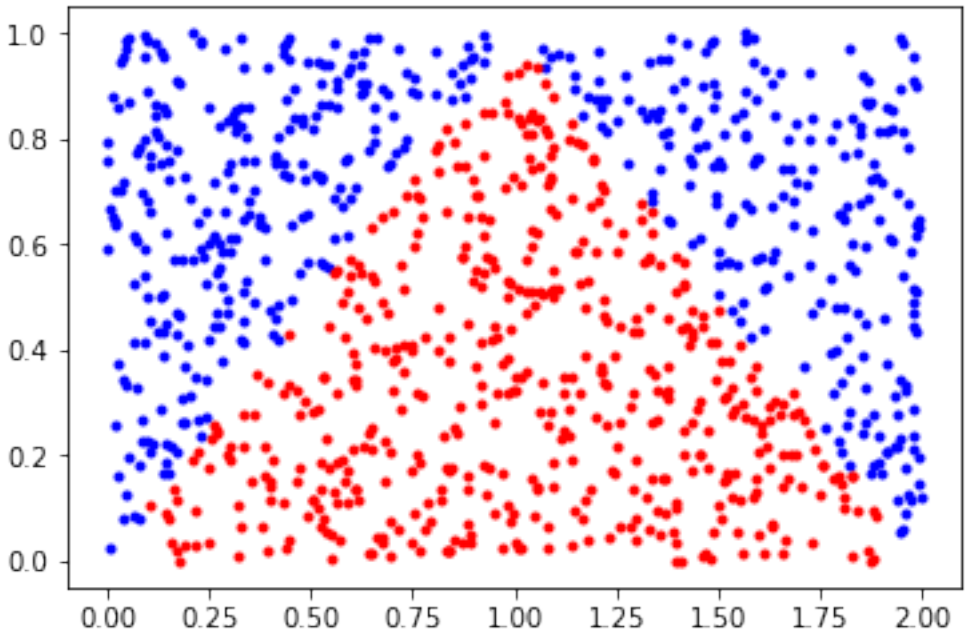
In [13]:
```
######################
#MONTECARLO_ITERATIVO
######################
```

In [14]:
```
%matplotlib inline
import time
import matplotlib.pyplot as plt
import random

def integra_mc(fun, a, b, M, num_puntos=1000):
    fig = plt.figure()
    dentro = 0
    #range->[0,1,2..num_puntos-1] equivale a for(int i=0; i<num_puntos; ++i)
    for i in range(num_puntos):
        #genera números aleatorios con distribución uniforme entre a y b
        x = random.uniform(a, b)
        y_simulado = random.uniform(0, M)
        #si el punto generado aleatoriamente esta dentro del área
        #(debajo de la linea)
        if y_simulado < fun(x):
            #incrementamos la cuenta de los puntos incluidos dentro del área
            dentro += 1
            plt.plot([x], [y_simulado], marker='o', markersize=3, color='red')
        else:
            plt.plot([x],[y_simulado], marker='o', markersize=3, color='blue')

    #fig.show()
    #fraccion multiplicado por la superficie del rectángulo
    return (dentro / num_puntos) * (b - a) * M
```

In [15]:
```
def fun(x): #función de un triangulo isósceles
    if x >= 0 and x < 1:
        return x
    elif x >= 1 and x < 2:
        return 2 - x
    else:
```

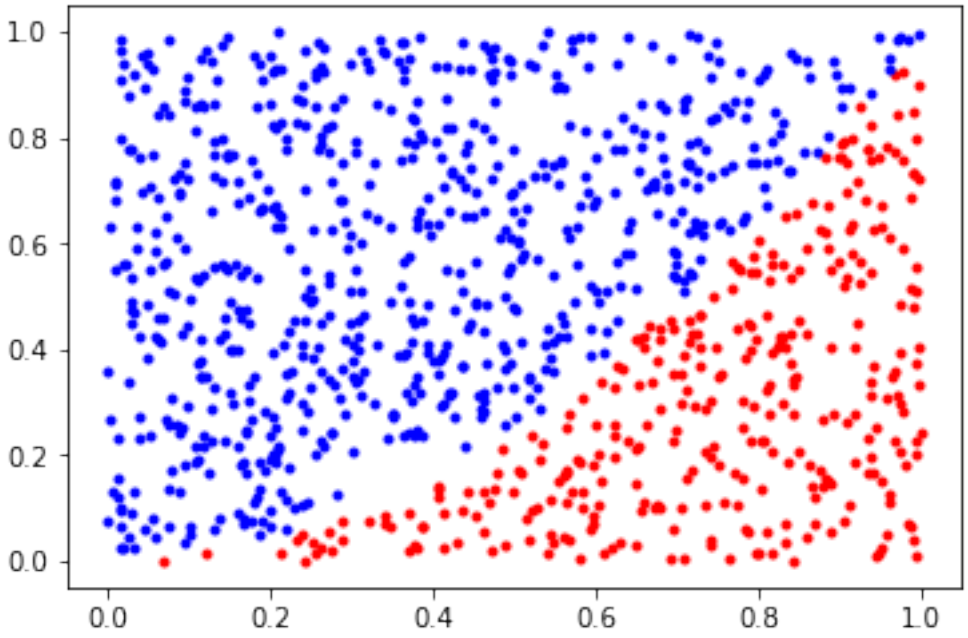
```
            return 0
        inicio = time.time()
        triangulo = integra_mc(fun, 0, 2, 1)
        fin = time.time()
        print('integral triangulo: {}, tiempo de ejecución: {}'
              .format(triangulo,fin-inicio))
```

```
integral triangulo: 0.996, tiempo de ejecución: 2.888279676437378
```




```
In [16]: def fun_cuadrado(x):#función cuadrática
             return x**2
         inicio = time.time()
         cuadrado = integra_mc(fun_cuadrado, 0, 1, 1)
         fin = time.time()
         print('integral cuadrado: {}, tiempo de ejecución: {}'
               .format(cuadrado,fin-inicio))
```

```
integral cuadrado: 0.323, tiempo de ejecución: 2.8414041996002197
```

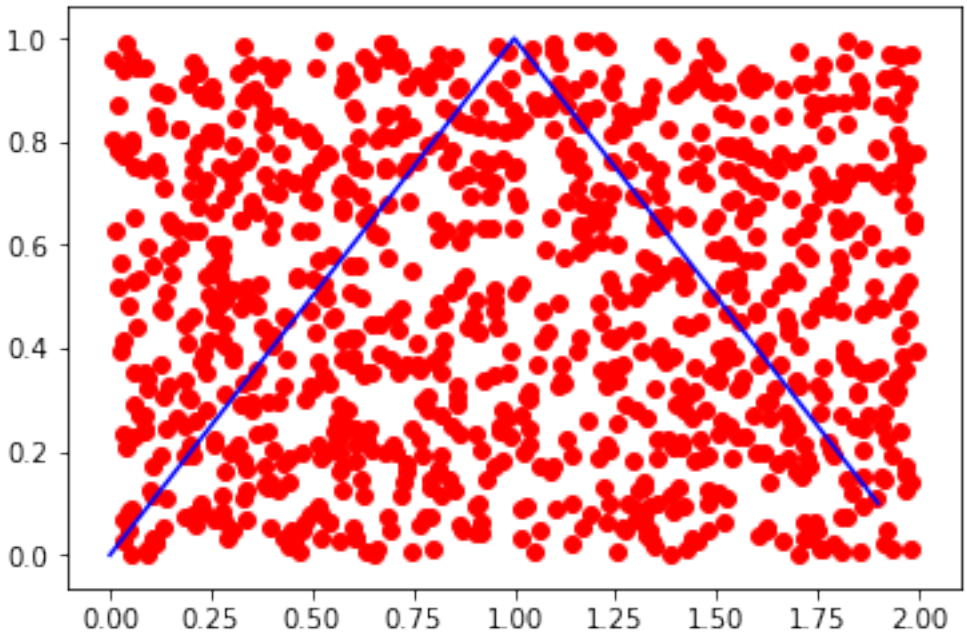


In [17]:
```
######################
#MONTECARLO_VECTORIAL
######################
```

In [18]:
```
%matplotlib inline
import time
import matplotlib.pyplot as plt
import numpy as np #operaciones con matrices, vectores

def integra_mc(fun, a, b, M, num_puntos=1000):
    fig = plt.figure()
    X = np.random.uniform(a, b, num_puntos)
    Y = [fun(x) for x in X]
    Y_simulado = np.random.uniform(0, M, num_puntos)

    plt.scatter(X, Y_simulado, marker='o', color='red')

    X_linea = np.arange(a, b, 0.1)
    Y_linea = [fun(x) for x in X_linea]
    plt.plot(X_linea, Y_linea, color='blue')

    esta_dentro = Y_simulado < Y
    dentro = sum(esta_dentro)
    total = len(esta_dentro)
    return (dentro / total) * (b - a) * M
```

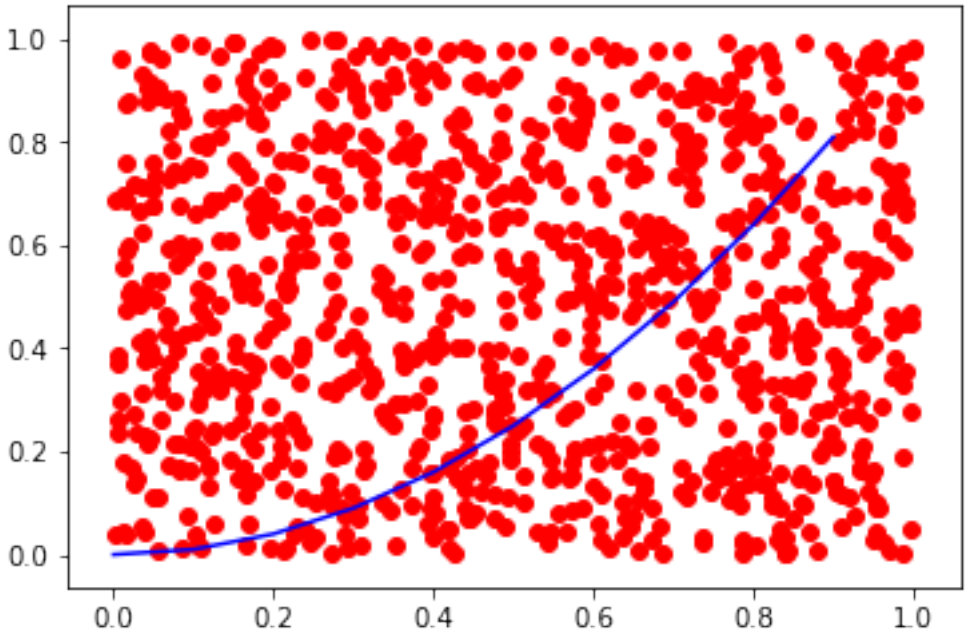
```
In [19]: def fun(x):
             if x >= 0 and x < 1:
                 return x
             elif x >= 1 and x < 2:
                 return 2 - x
             else:
                 return 0
         inicio = time.time()
         triangulo = integra_mc(fun, 0, 2, 1)
         fin = time.time()
         print('integral triángulo: {}, tiempo de ejecución: {}'
               .format(triangulo,fin-inicio))
```

```
integral triángulo: 0.98, tiempo de ejecución: 0.0468745231628418
```




```
In [20]: def fun_cuadrado(x):
             return x**2

         inicio = time.time()
         cuadrado = integra_mc(fun_cuadrado, 0, 1, 1)
         fin = time.time()
         print('integral cuadrado: {}, tiempo de ejecución: {}'
               .format(cuadrado,fin-inicio))
```

```
integral cuadrado: 0.335, tiempo de ejecución: 0.11070370674133301
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0