# Procesamiento y Análisis de Datos Astronómicos

## 3.- Números Aleatorios

R. Gil-Hutton

Marzo 2020

# Actividades:

## Práctica 2:

- Leer los datos del archivo elegido en la práctica anterior, graficar una columna cualquiera contra otra agregando nombre a los ejes (por ejemplo, Vmag vs. distancia).
- Sobre la figura anterior, graficar en otro color lo mismo pero seleccionando solo los casos que cumplan una cierta condición usando datos de una tercera columna (por ejemplo, Vmag vs. distancia pero para $(B - V) > 0{,}5$).
- Si quisiera representar la primera figura en un histograma bidimensional, cómo lo puedo hacer con matplotlib (ver ejemplo)?

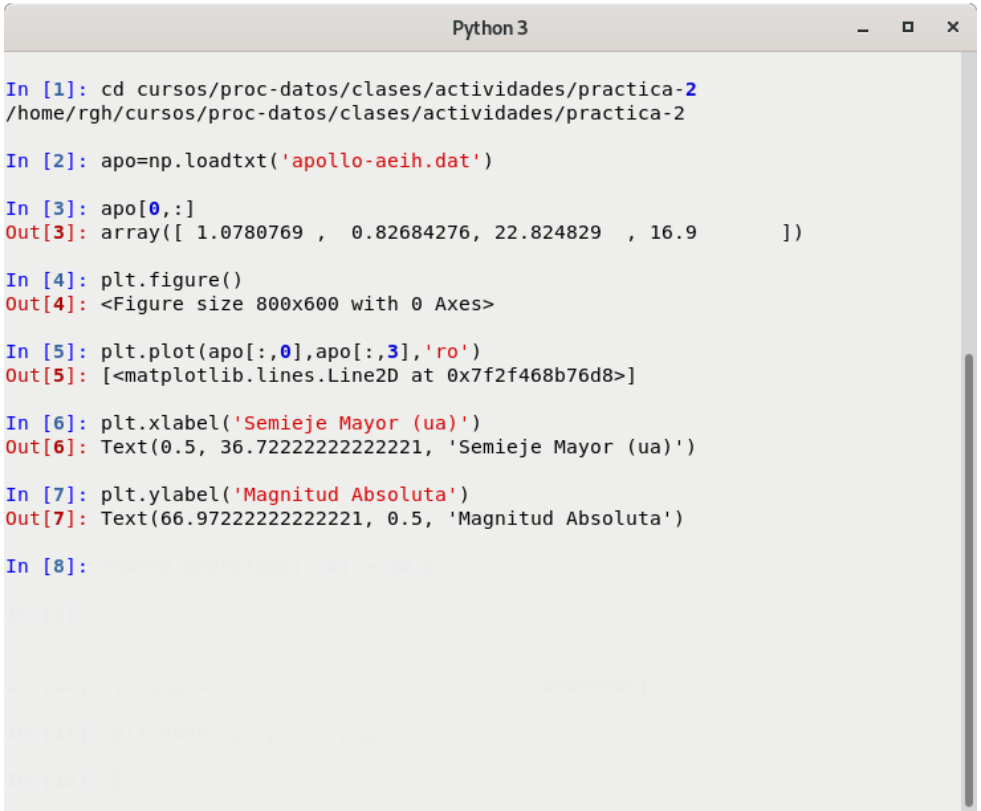
# Actividades:

## Práctica 2 (cont.):

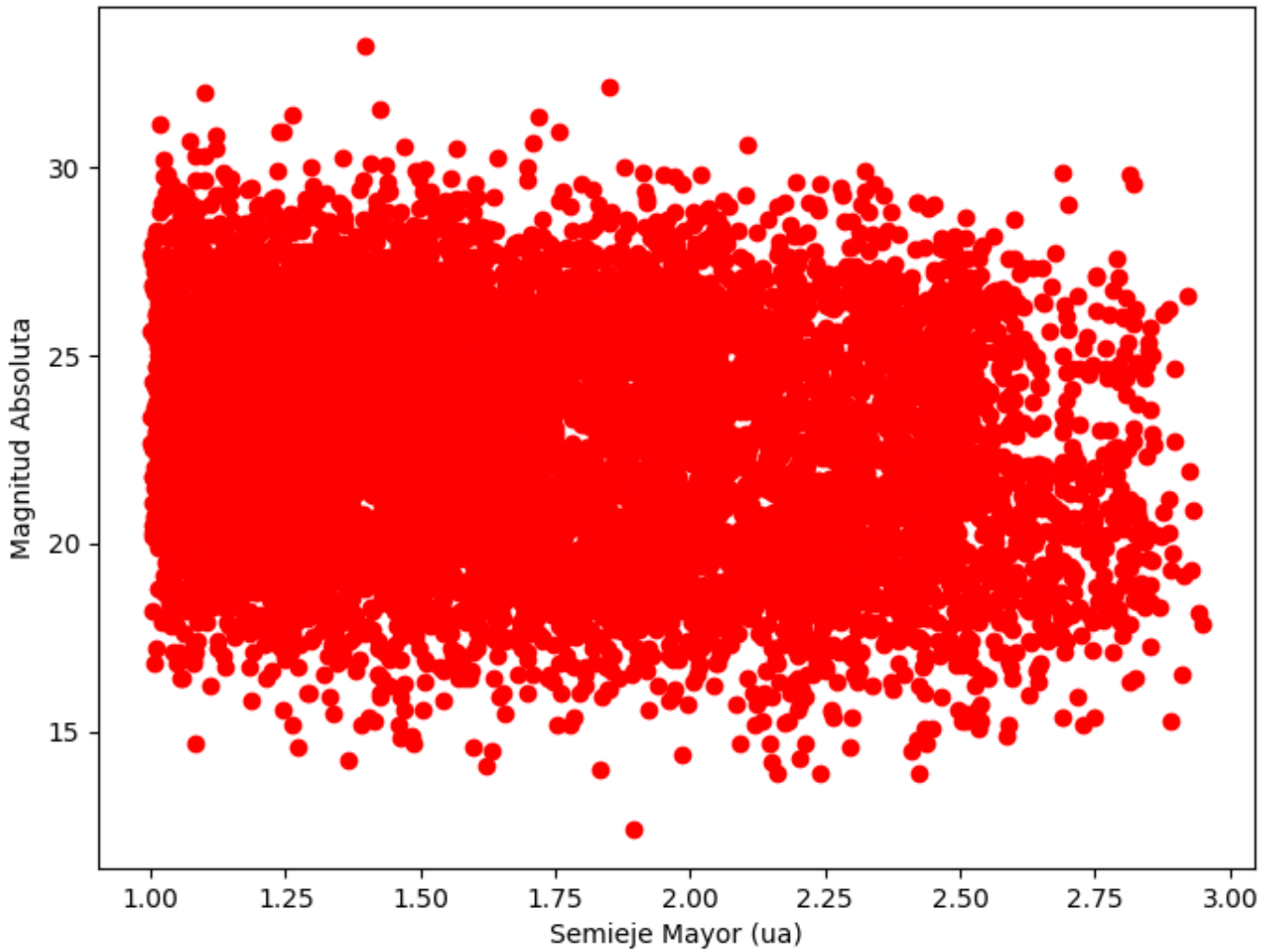
- para hacer un histograma, cuáles son las diferencias entre usar plt.bar() y plt.hist()?.
- Cuántos colores hay disponibles en plt.plot() para graficar?
- Cuántos tipos de líneas y marcadores hay disponibles en plt.plot() para graficar?

# Actividades:

Como ejemplo de archivo usaré **apollo-aeih.dat** que lista semieje mayor, excentricidad,inclinación y magnitud absoluta para un grupo de asteroides Apollo. Grafico semieje vs. magnitud absoluta:



# Actividades:

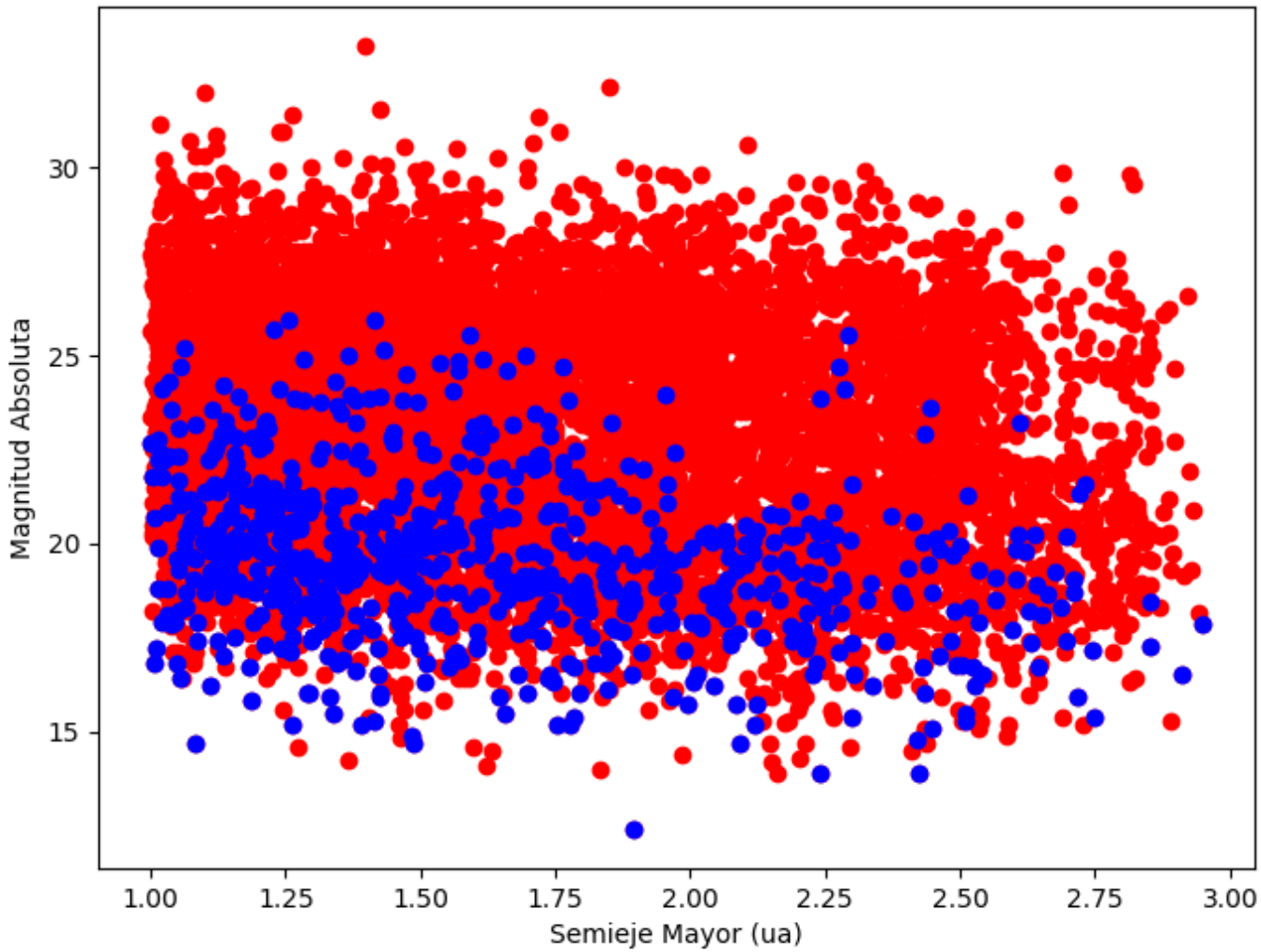


# Actividades:

Ahora grafico semieje vs. magnitud absoluta pero para los que tienen inclinación mayor de 30 grados:

```
In [1]: cd cursos/proc-datos/clases/actividades/practica-2
/home/rgh/cursos/proc-datos/clases/actividades/practica-2

In [2]: apo=np.loadtxt('apollo-aeih.dat')

In [3]: apo[0,:]
Out[3]: array([ 1.0780769 ,  0.82684276, 22.824829  , 16.9       ])

In [4]: plt.figure()
Out[4]: <Figure size 800x600 with 0 Axes>

In [5]: plt.plot(apo[:,0],apo[:,3],'ro')
Out[5]: [<matplotlib.lines.Line2D at 0x7fc7bc9d2550>]

In [6]: plt.xlabel('Semieje mayor (ua)')
Out[6]: Text(0.5, 36.72222222222221, 'Semieje mayor (ua)')

In [7]: plt.ylabel('Magnitud Absoluta')
Out[7]: Text(66.97222222222221, 0.5, 'Magnitud Absoluta')

In [8]: inx=np.where(apo[:,2] > 30.)

In [9]: altai=apo[inx]

In [10]: plt.plot(altai[:,0],altai[:,3],'bo')
Out[10]: [<matplotlib.lines.Line2D at 0x7fc7b5511748>]

In [11]:
```

# Actividades:

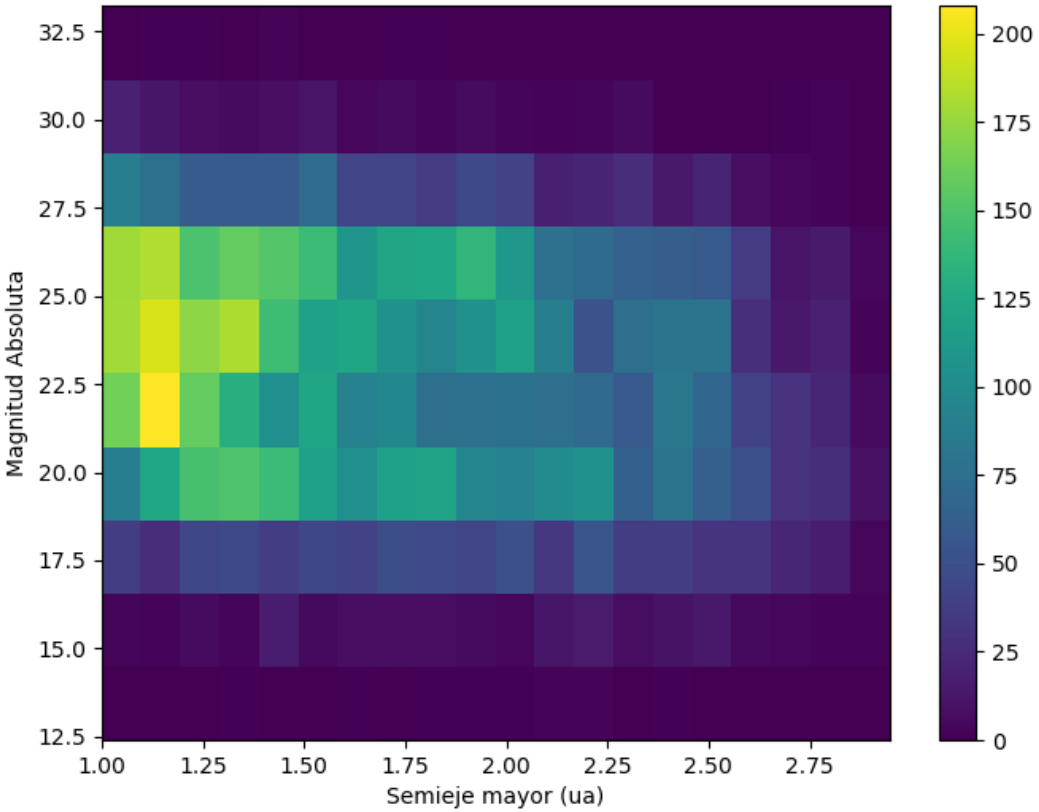


# Actividades:

Pra graficar el histograma en dos dimensiones de la primera figura se puede usar plt.hist2d:

```
In [24]: plt.clf()

In [25]: his=plt.hist2d(apo[:,0],apo[:,3],bins=[20,10])

In [26]: plt.colorbar()
Out[26]: <matplotlib.colorbar.Colorbar at 0x7f087a8b4a58>

In [27]: plt.xlabel('Semieje mayor (ua)')
Out[27]: Text(0.5, 36.72222222222221, 'Semieje mayor (ua)')

In [28]: plt.ylabel('Magnitud Absoluta')
Out[28]: Text(53.847222222222214, 0.5, 'Magnitud Absoluta')

In [29]:
```

# Actividades:



# Actividades:

- plt.bar() hace un gráfico de barras sobre datos ya obtenidos (por ejemplo con np.histogram()) mientras que plt.hist() calcula y dibuja directamente el histograma desde un conjunto de datos.
- En plt.plot() hay disponibles 256*256*256=16777216 colores (parámetros: color, markerfacecolor, o mfc, y markeredgecolor, o mec).
- En plt.plot() hay disponibles 5 tipos de líneas básicas (parámetros: linestyle, o ls, y linewidth, o lw), incluyendo sin línea (None), pero las líneas cortadas también se pueden configurar (parámetro: dashes).
- En plt.plot() hay disponibles 22 tipos de marcadores (parámetros: marker, markersize, o ms, y markeredgewidth, o mew).

# Números aleatorios:

- Necesarios para cualquier simulación o estimación de parámetros.
- El objetivo es generar números entre dos valores dados con un período de repetición infinito.
- En cualquier aplicación computacional el limitante es el modo de representación de números utilizado.
- en general dependen de la capacidad del procesador utilizado y usualmente son de mala calidad.
- Una buena explicación está en "Numerical Recipes", Press et al.

# Números aleatorios:

- Dependen de la distribución asumida a priori.
- En realidad son números pseudo-aleatorios.
- Todos los lenguajes de computación tienen generadores de números aleatorios para diferentes distribuciones (uniforme, gaussiana, exponencial, Poisson, etc.)
- Independientemente de la distribución asumida, todos los generadores se basan en un generador con distribución uniforme basado en un algoritmo conocido.

# Generador uniforme:

- generadores lineales multiplicativos congruentes:

$$I_{(j+1)} = \quad \text{mód}\ _m(a * I_j + c) \ \ paraj = 1, 2, ...$$

  m es el módulo, a el multiplicador y c el incremento. Este es el algoritmo de Lehmer (1951).
- como se utiliza recurrencia, la serie de números generados se repite a si misma con un período no mayor a m.

# Generador uniforme:

- m debe ser un entero primo suficientemente grande.
- a debe ser un entero entre 2 y m-1.
- el valor inicial o semilla, $I_1$ , debe ser un entero entre 1 y m-1.
- se obtienen valores reales en $(0, 1)$ haciendo:

$$u = (I/m) - int(I/m).$$

- como m es primo, I no puede ser 0.
- u no puede ser 0 o 1.
- el menor valor posible de u es $(1/m)$.
- el mayor valor posible de u es $(1 - 1/m)$.

# Generador uniforme:

Ejemplos con semilla diferente:

- con m=13, a=6, c=0; $I_1$=1 la serie es:

  1,6,10,8,9,2,12,7,3,5,4,11,1,...

- con m=13, a=6, c=0; $I_1$=2 la serie es:

  2,12,7,3,5,4,11,1,6,10,8,9,2,...

El cambio de semilla desplaza la serie

# Generador uniforme:

Ejemplos con multiplicador malo:

- con m=13, a=5, c=0; $I_1$=1 la serie es:

  1,5,12,8,1,...

- con m=13, a=5, c=0; $I_1$=2 la serie es:

  2,10,11,3,2,...

Se debe elegir un multiplicador correcto para asegurar una serie extensa (para m=13, a=2,6,7, y 11)

# Generador uniforme:

- para un procesador de 32 bits el mayor entero posible es $2^{31}$.
- el mayor número primo conocido hasta hoy es:

$$2^{31} - 1 = 2147483647.$$

- un multiplicador que nos da la serie completa para este valor es $7^5 = 16807$, pero hay varios cientos de millones de posibles valores (48271 y 69621 son otras dos posibilidades).
- en este caso, c $= 0$ es suficiente.
- trabajando en 32 bits pueden aparecer problemas de overflow cuando hacemos $(a * I_j)$.

# Números aleatorios en Python:

- Python posee generadores de números aleatorios que corresponden a distribuciones conocidas. Los más usuales se encuentran en numpy en su módulo random, pero existen muchos más en el módulo stats de scipy.
- Con las funciones disponibles en numpy es posible obtener números aleatorios a partir de diferentes distribuciones, tales como beta, binomial, chi cuadrado, exponencial, gamma, hipergeométrica, geométrica, log-normal, normal, poisson, en ley de potencias, t de Student, etc..

# Números aleatorios en Python:

- Un listado de las funciones disponibles en el módulo random se obtienen haciendo un help del módulo.
- La función np.random.random_sample() (o su alias np.random.random()) permite obtener números aleatorios reales en el rango [0,0, 1,0) tomados de una distribución uniforme.
- La función np.random.standard_normal() permite obtener números aleatorios tomados de una distribución normal con media 0 y varianza 1, $N(0, 1)$. Para otra media y varianza hay que hacer:
  $N(\mu, \sigma^2) = \sigma \times np.random.standard\_normal() + \mu$.
- También en el módulo random se encuentra la función np.random.seed() que permite inicializar los generadores de números aleatorios cambiando la semilla.

# Números aleatorios en Python:

```
In [12]: np.random.random()
Out[12]: 0.48759755232850643

In [13]: np.random.random(5)
Out[13]: array([0.26011413, 0.06109245, 0.10037846, 0.14655373, 0.30602375])

In [14]: np.random.random((2,3))
Out[14]:
array([[0.82189697, 0.40549623, 0.98760054],
       [0.38907536, 0.23929474, 0.76810194]])

In [15]: np.random.standard_normal()
Out[15]: -0.4623806596250095

In [16]: np.random.standard_normal((2,3))
Out[16]:
array([[ 0.19760422, -0.06575084,  0.92800172],
       [-1.61927564, -0.55628399, -0.0033318 ]])

In [17]:
```

# Números aleatorios en Python:

Hay que tener algo de cuidado cuando se usa la función np.random.seed() porque se puede perder la aleatoriedad buscada:

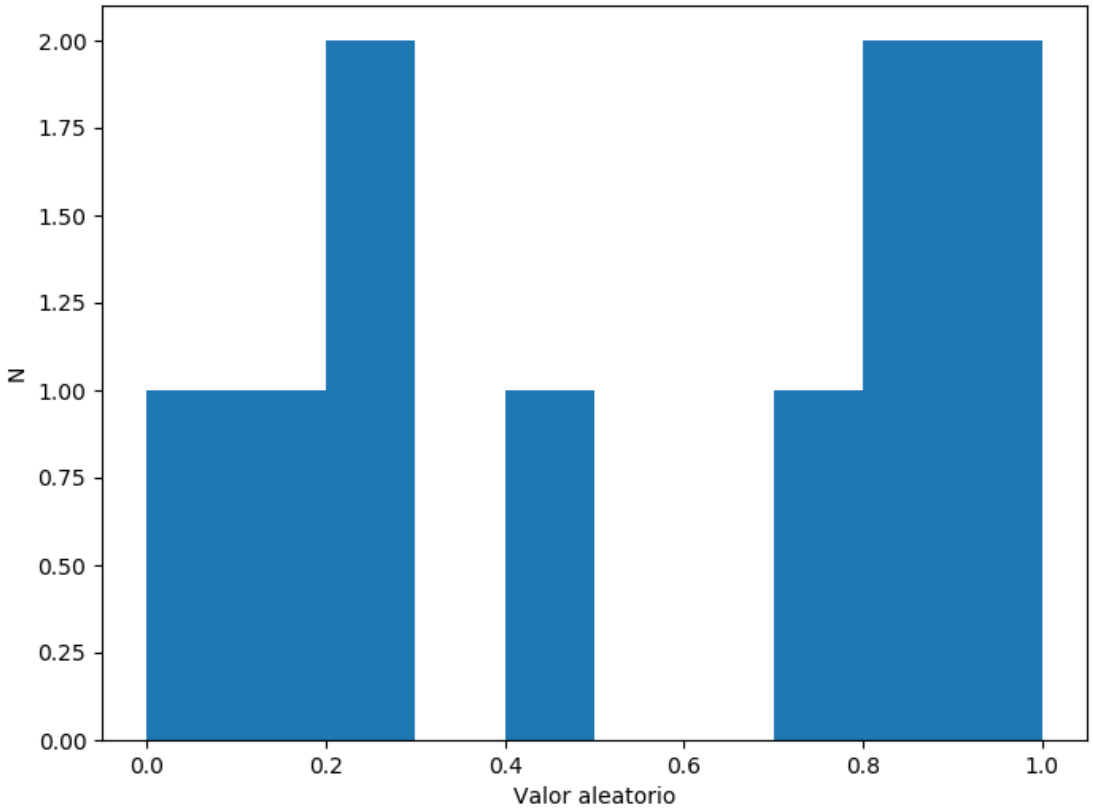
```
In [18]: np.random.seed(0)

In [19]: np.random.random(5)
Out[19]: array([0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ])

In [20]: np.random.seed(0)

In [21]: np.random.random(5)
Out[21]: array([0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ])

In [22]:
```

# Números aleatorios en Python:

Por otra parte, la aleatoriedad se obtiene siempre que la muestra utilizada sea grande o, lo que es lo mismo, hay que ser cuidadoso con las muestras pequeñas:

```
In [150]: hh=np.random.random(10)

In [151]: hh
Out[151]:
array([0.49106216, 0.97393847, 0.77721517, 0.12402262, 0.87059602,
       0.99820345, 0.20450379, 0.08120062, 0.25931946, 0.86615069])

In [152]: his=np.histogram(hh,bins=10,range=(0,1))

In [153]: his
Out[153]:
(array([1, 1, 2, 0, 1, 0, 0, 1, 2, 2]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]))

In [154]: plt.clf()

In [155]: plt.bar(his[1][1:]-0.05,his[0],0.1)
Out[155]: <BarContainer object of 10 artists>

In [156]:
```

# Números aleatorios en Python:

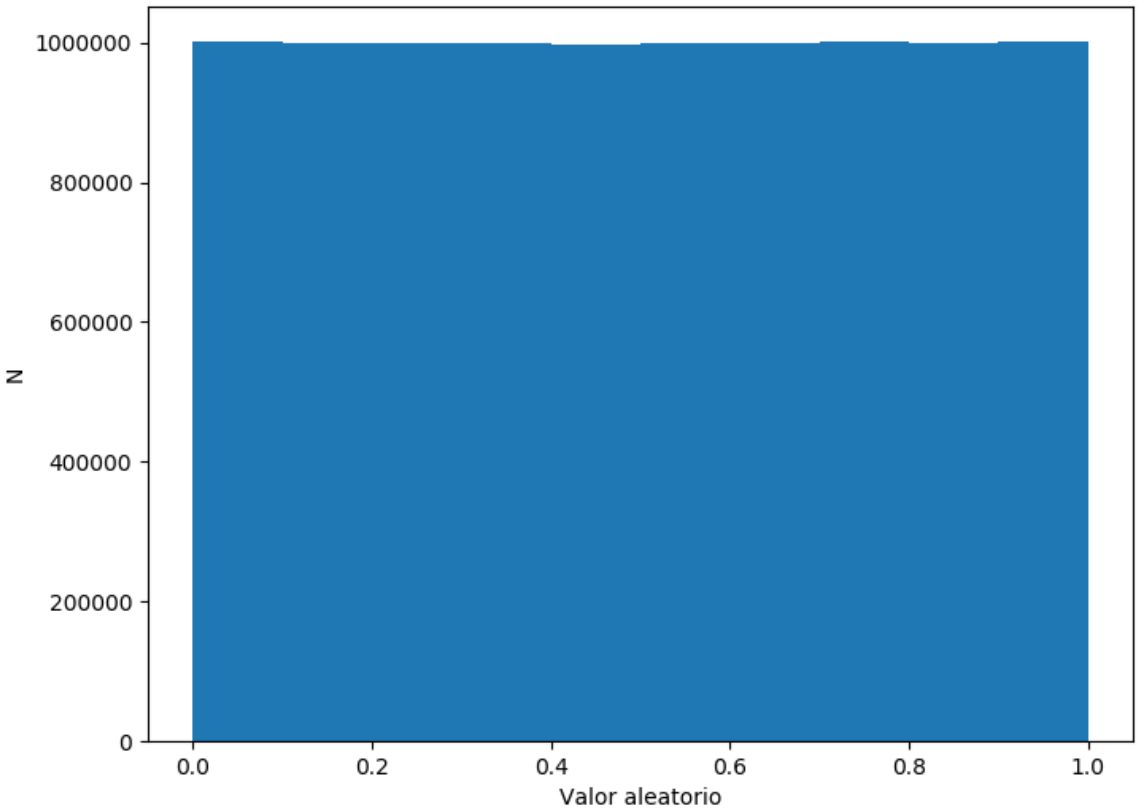


# Números aleatorios en Python:

Si incrementamos la muestra significativamente:

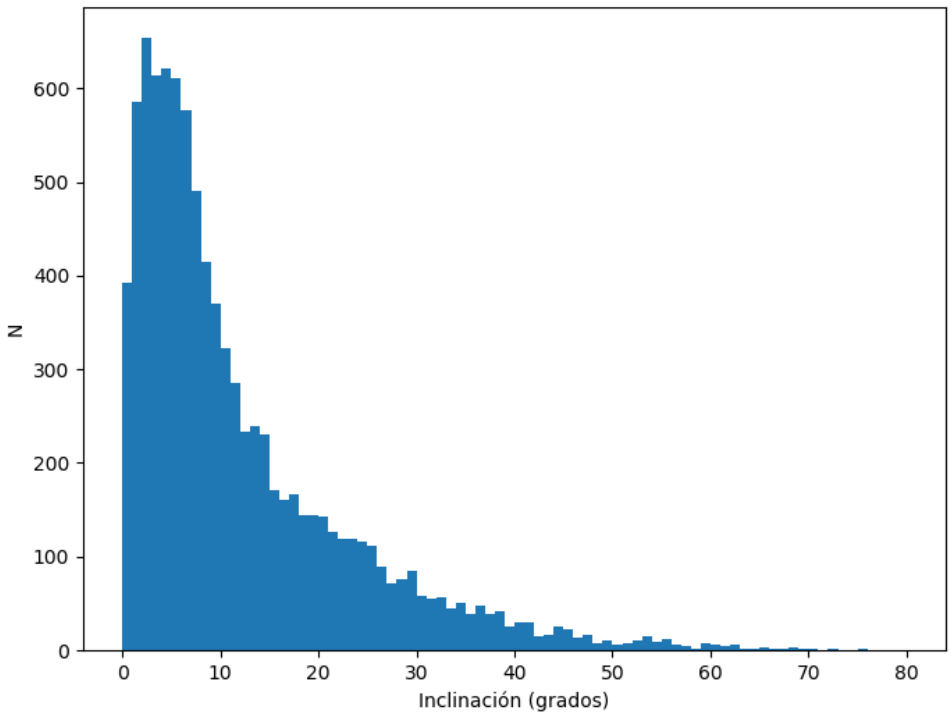
```
In [162]: hh=np.random.random(10000000)

In [163]: his=np.histogram(hh,bins=10,range=(0,1))

In [164]: plt.clf()

In [165]: plt.bar(his[1][1:]-0.05,his[0],0.1)
Out[165]: <BarContainer object of 10 artists>

In [166]:
```

# Números aleatorios en Python:



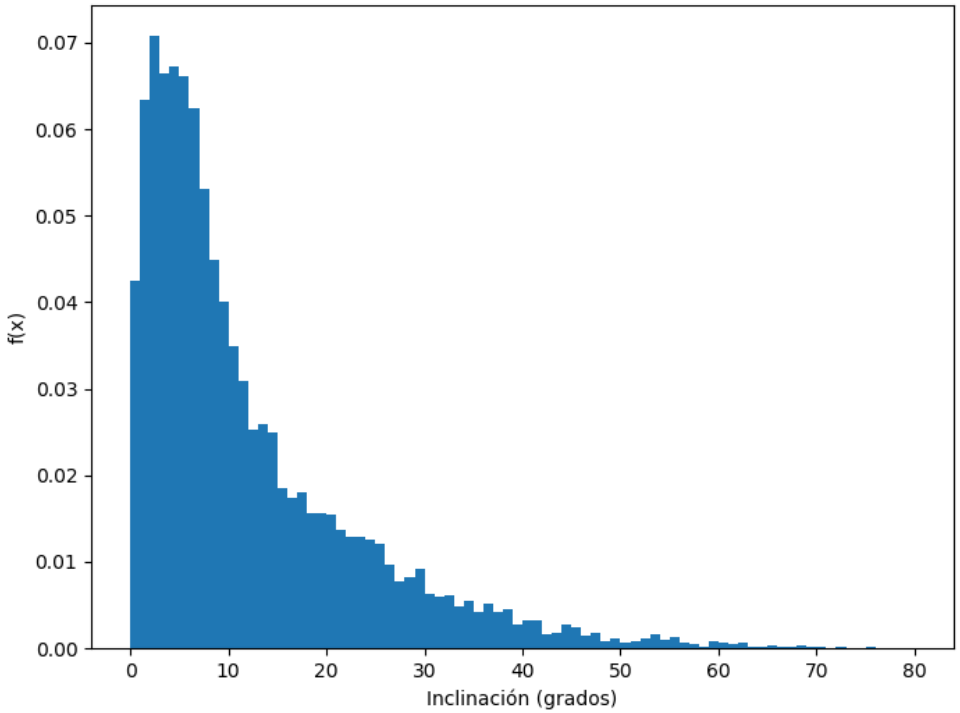
# Distribuciones arbitrarias:

Supongamos que queremos generar números aleatorios a partir de la distribución de una cierta población para lo cual se toma una muestra aleatoria que se asume representativa de la población general. Por ejemplo, tomemos para trabajar el histograma de las inclinaciones de los objetos Apollo:



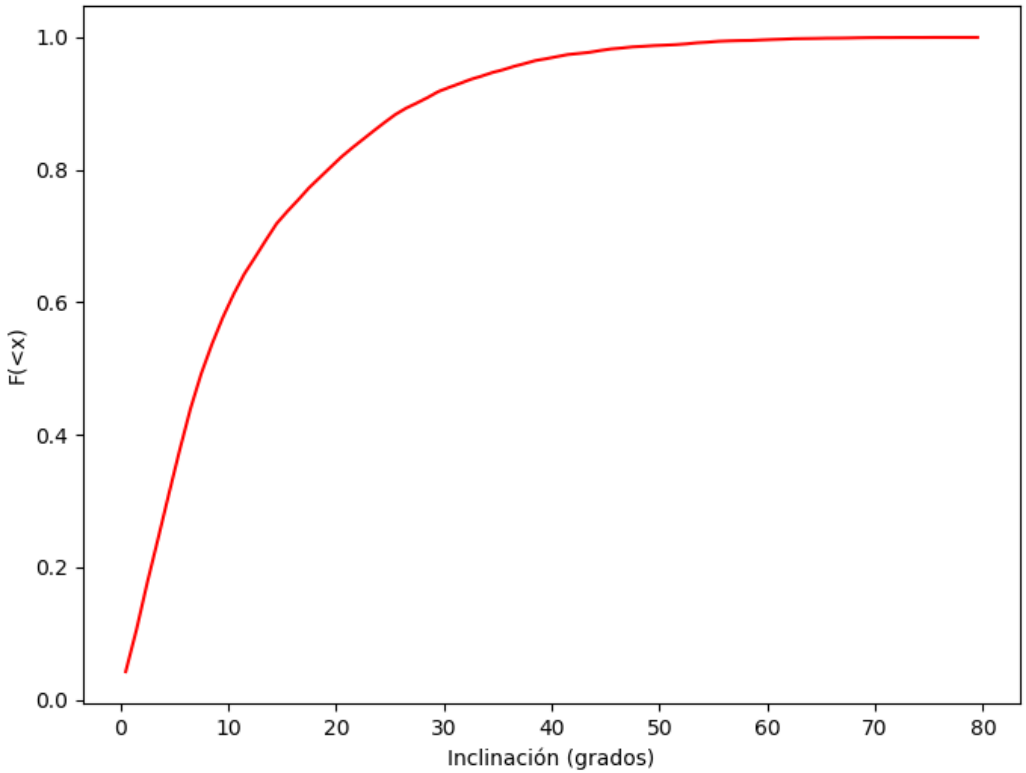
# Distribuciones arbitrarias:

Primero hay que obtener una distribución de probabilidad al convertir este histograma en un histograma de frecuencia relativa normalizando el área del histograma a un valor de 1:



# Distribuciones arbitrarias:

Luego, calcular la frecuencia relativa acumulada de la muestra, $F(< x) = f(< x)/N$. Finalmente, como $F(< x)$ es siempre creciente, con un generador uniforme obtener un valor de $F(< x)$ y averiguar a que valor de $x$ corresponde.



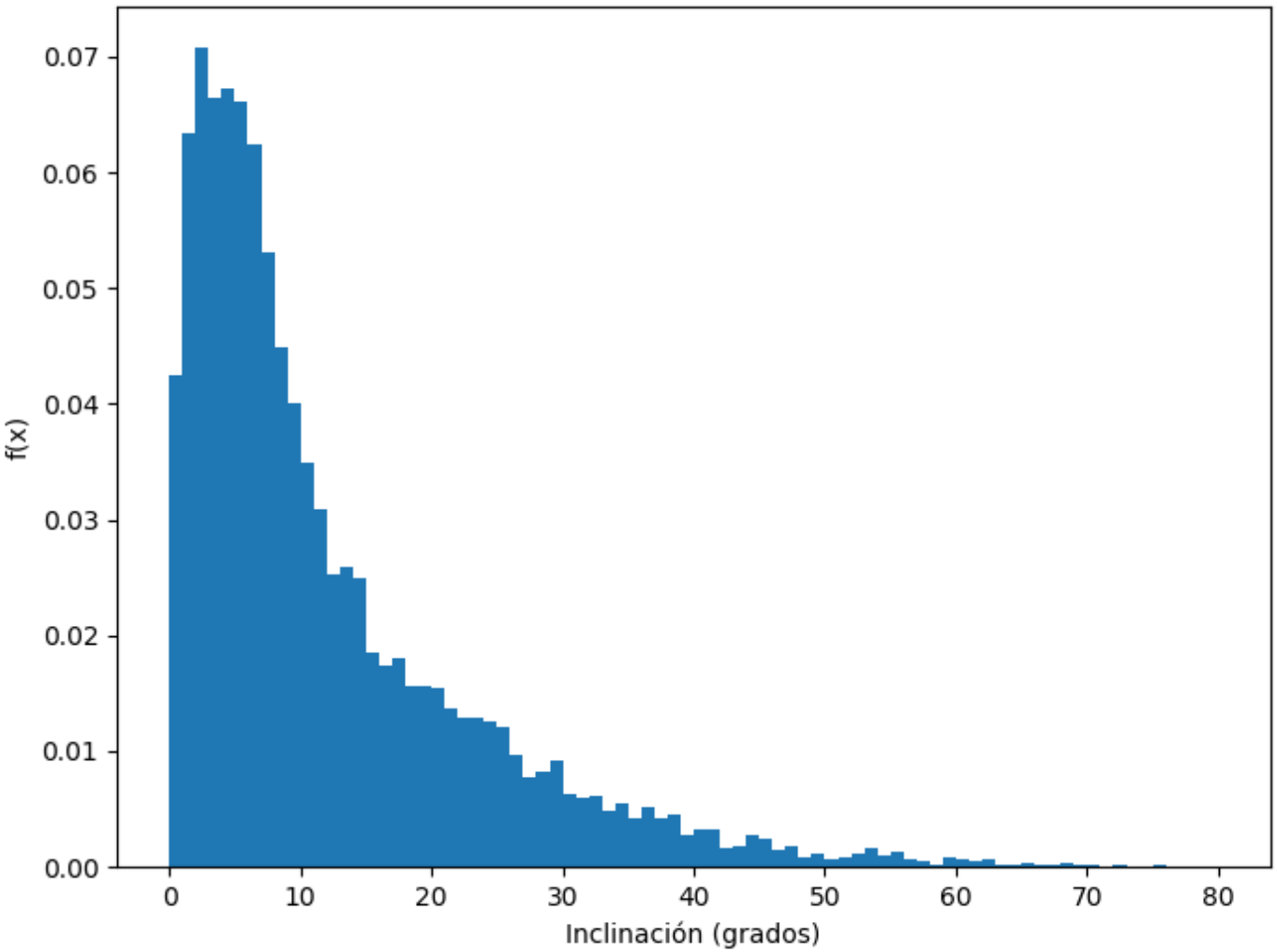
# Distribuciones arbitrarias:

```
In [109]: frec=np.histogram(apo[:,2],bins=80,range=(0,80),density=True)

In [110]: frec
Out[110]:
(array([0.04253707, 0.06342678, 0.07078688, 0.06645741, 0.0673233 ,
        0.0661327 , 0.06234441, 0.05314428, 0.04481004, 0.04004762,
        0.03496049, 0.03095573, 0.02521918, 0.0258686 , 0.02500271,
        0.0185085 , 0.01731789, 0.01807555, 0.0155861 , 0.0155861 ,
        0.01547787, 0.01363784, 0.01288018, 0.01288018, 0.01255547,
        0.01201429, 0.00963308, 0.00779305, 0.008226  , 0.00909189,
        0.00627774, 0.00595303, 0.0061695 , 0.00487066, 0.00541184,
        0.00422124, 0.00508713, 0.00422124, 0.00454595, 0.00281416,
        0.00313887, 0.00313887, 0.00162355, 0.00173179, 0.00270592,
        0.00238121, 0.00140708, 0.00173179, 0.00086589, 0.00108237,
        0.00064942, 0.00075766, 0.00119061, 0.00162355, 0.00097413,
        0.00129884, 0.00064942, 0.00043295, 0.00021647, 0.00075766,
        0.00064942, 0.00054118, 0.00064942, 0.00021647, 0.00021647,
        0.00032471, 0.00010824, 0.00021647, 0.00032471, 0.00021647,
        0.00010824, 0.        , 0.00010824, 0.        , 0.        ,
        0.00010824, 0.        , 0.        , 0.        , 0.        ]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
        13., 14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25.,
        26., 27., 28., 29., 30., 31., 32., 33., 34., 35., 36., 37., 38.,
        39., 40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50., 51.,
        52., 53., 54., 55., 56., 57., 58., 59., 60., 61., 62., 63., 64.,
        65., 66., 67., 68., 69., 70., 71., 72., 73., 74., 75., 76., 77.,
        78., 79., 80.]))

In [111]: acu=np.copy(frec[0])

In [112]: acu=np.cumsum(acu,dtype=float)

In [113]: plt.clf()

In [114]: plt.plot(his[1][1:]-0.5,acu,'-r')
Out[114]: [<matplotlib.lines.Line2D at 0x7f8d503c2400>]

In [115]:
```

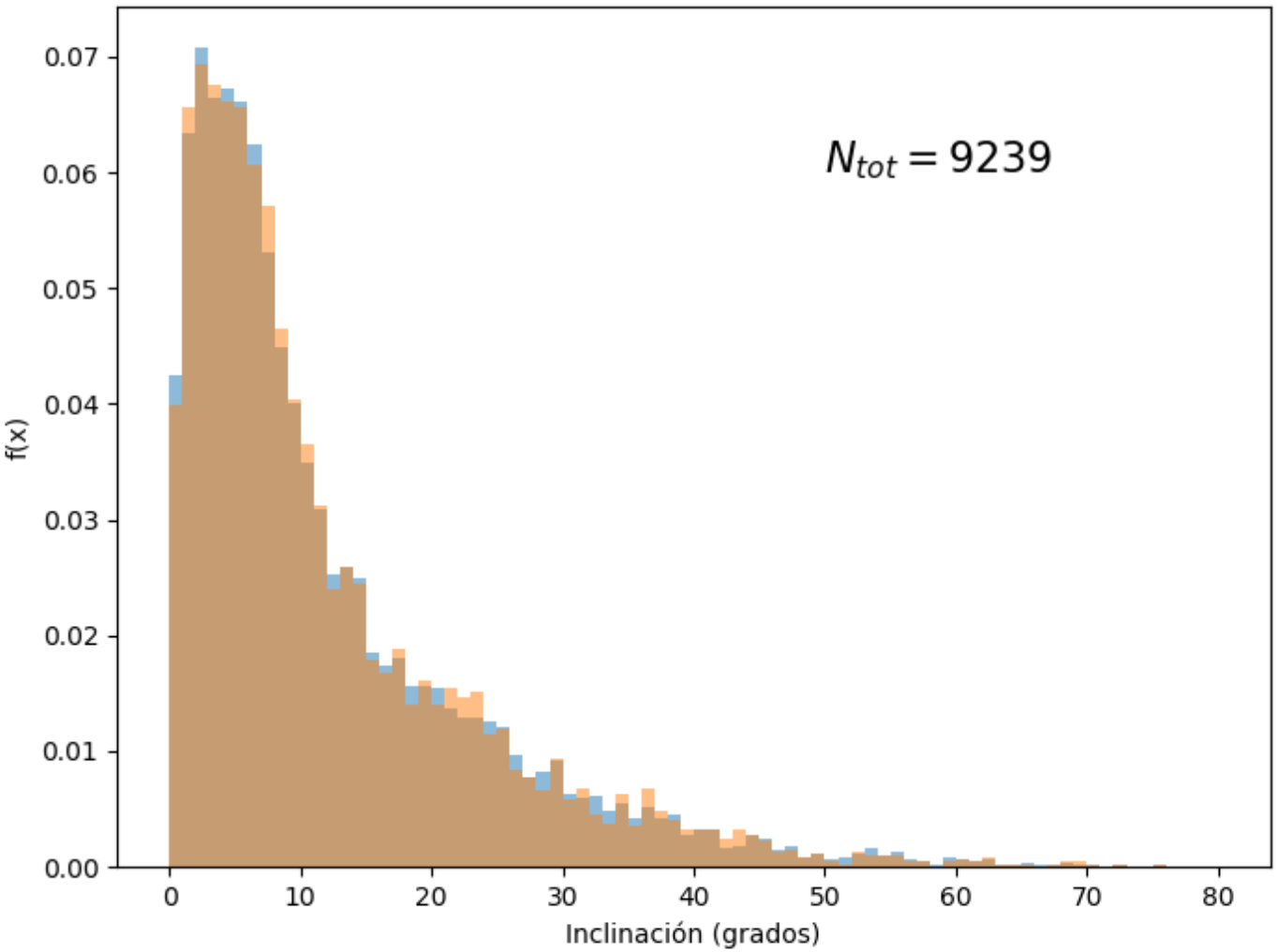
# Distribuciones arbitrarias:

```
In [116]: np.sum(his[0])
Out[116]: 9239

In [117]: rr=np.random.random(9239)

In [118]: val=np.zeros((len(rr)))

In [119]: for ii in range(len(rr)):
     ...:     # encuentra a que bin corresponde
     ...:     #
     ...:     jj=0
     ...:     while(rr[ii] > acu[jj]):
     ...:         jj+=1
     ...:     #conocido el bin toma al azar un valor en el
     ...:     #
     ...:     val[ii]=his[1][jj]+0.5+(np.random.random()-0.5)
     ...:

In [120]: sim=np.histogram(val,bins=80,range=(0,80),density=True)

In [121]: plt.clf()

In [122]: plt.bar(frec[1][1:]-0.5,frec[0],1,alpha=0.5)
Out[122]: <BarContainer object of 80 artists>

In [123]: plt.bar(sim[1][1:]-0.5,sim[0],1,alpha=0.5)
Out[123]: <BarContainer object of 80 artists>

In [124]:
```

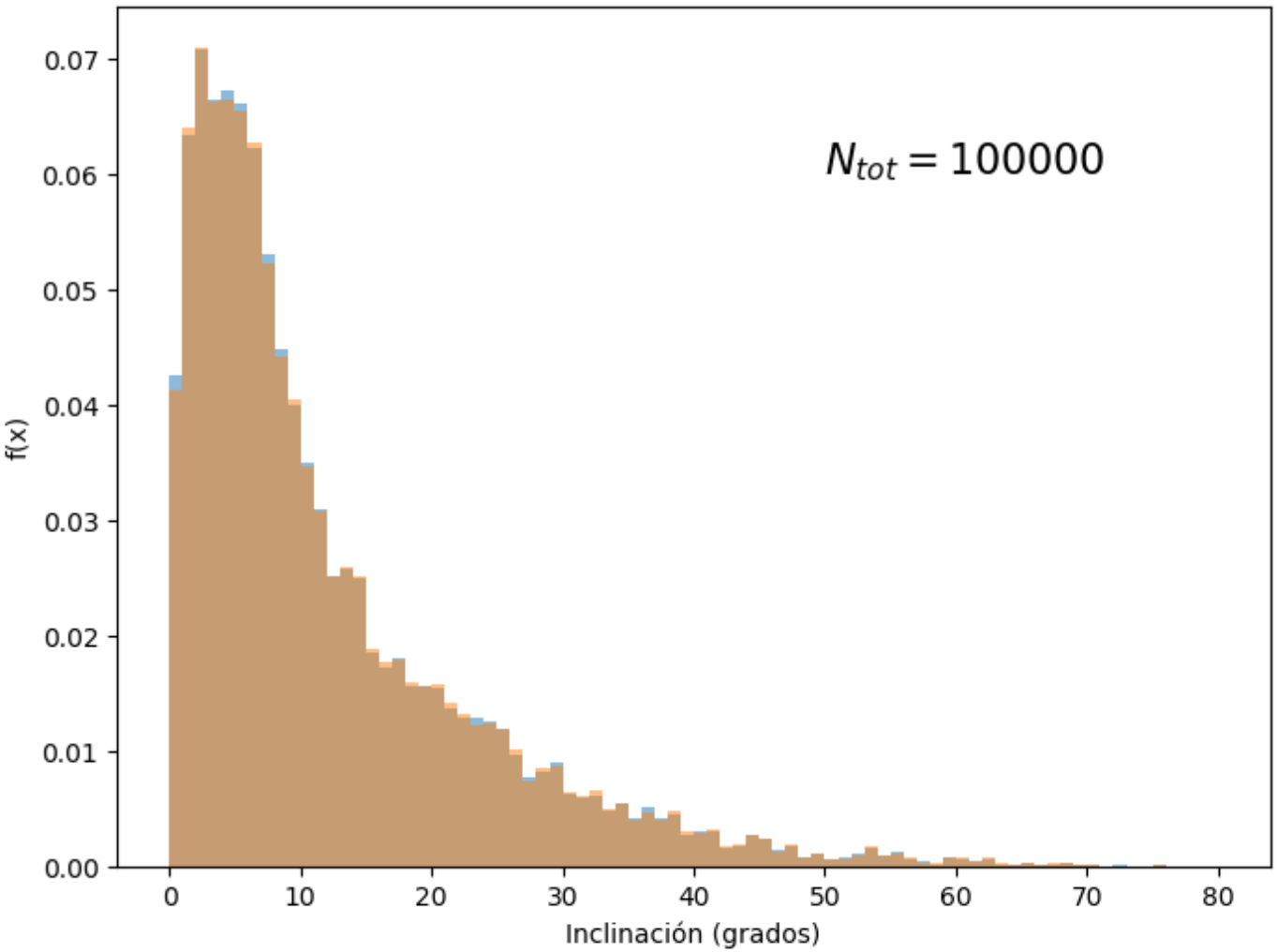
# Distribuciones arbitrarias:



# Distribuciones arbitrarias:



# Distribuciones arbitrarias:



# Funciones:

## Funciones utilizadas:

| | | | |
|---|---|---|---|
| plt.figure() | plt.plot() | plt.xlabel() | plt.ylabel() |
| plt.savefig() | plt.clf() | plt.bar() | np.where() |
| np.zeros() | np.histogram() | np.copy() | np.sum() |
| np.random.random() | | np.random.rand() | |

## Funciones útiles:

| | | | |
|---|---|---|---|
| plt.title() | plt.xticks() | plt.yticks() | plt.sub.plot() |
| plt.legend() | plt.annotate() | plt.scatter() | plt.imshow() |
| plt.contour() | plt.errorbar() | plt.hist() | plt.polar() |
| plt.pie() | np.min() | np.max() | np.mgrid() |

# Actividades:

## Práctica 3:

- Extraer del archivo de datos los valores para una de las variables y graficar su histograma, simular la distribución obtenida con una muestra de 300000 valores, y comparar los histogramas de frecuencias.
- Generar $10^8$ pares de valores $(X, Y)$ utilizando un generador de números aleatorios uniforme. Verificar que si el número de casos que cumplen con $X^2 + Y^2 \leq 1$ es $M$, se obtiene que $4 \times M/10^8 \approx \pi$.

## Entrega

Para la próxima clase

Por consultas: ricardo.gil-hutton@conicet.gov.ar
Grupo de Ciencias Planetarias - CUIM 2