Ejercicios semana 19 de marzo - Alejandro Santorum

March 25, 2018

In [36]:

Sistema Fisico

In [89]:
```
def simulacion(L):
    i=2;j=2
    while(i==j):
        i = randint(0, len(L)-1)
        j = randint(0, len(L)-1)
    if(L[i]>0):
        L[i] = L[i] - 1
        L[j] = L[j] + 1
    return L
```

In [37]:
```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in xsrange(10):
    print simulacion(L)
```

```
[2, 2, 3, 4, 5, 6, 7, 8, 8]
[2, 2, 3, 4, 4, 6, 7, 8, 9]
[1, 2, 4, 4, 4, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 2, 4, 5, 6, 7, 8, 10]
[2, 2, 2, 3, 5, 6, 7, 8, 10]
[2, 2, 3, 3, 5, 6, 7, 7, 10]
[2, 1, 3, 3, 5, 6, 7, 8, 10]
[2, 2, 3, 3, 5, 6, 6, 8, 10]
[2, 2, 3, 3, 4, 6, 6, 9, 10]
```

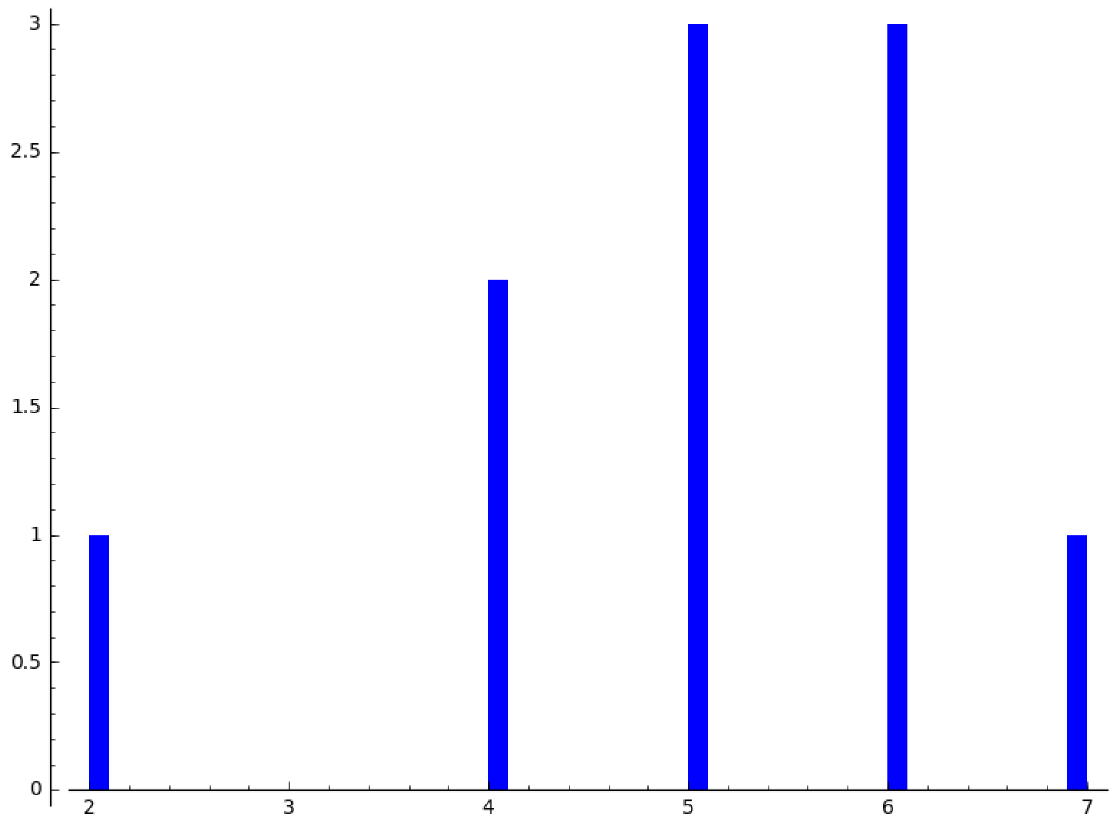
In [63]:
```
def evolFisica(n, N):
    L = list()
    for i in xsrange(n):
        L.append(5)
    for j in xsrange(N):
        L = simulacion(L)
    return L
```

In [64]:
```
L = evolFisica(10, 10)
print L
print (sum(L)/len(L)).n(digits = 3)
```

```
[6, 4, 2, 6, 6, 4, 5, 7, 5, 5]
5.00
```

In [65]:
```
T = finance.TimeSeries(L)
T.plot_histogram()
```

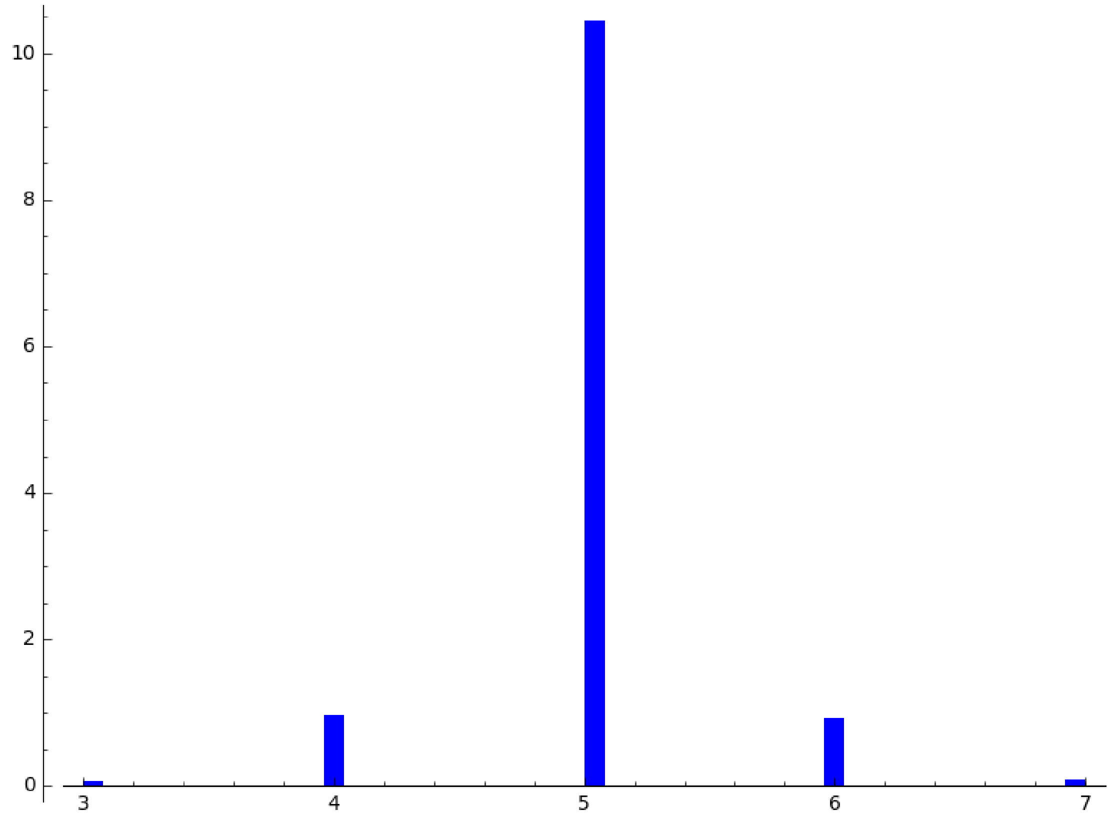
Out[65]:



In [82]:
```
listaHistogramas = list()
for k in xsrange(5):
    AUX = evolFisica(1000, 100*(10**k))
    listaHistogramas.append(AUX)
```

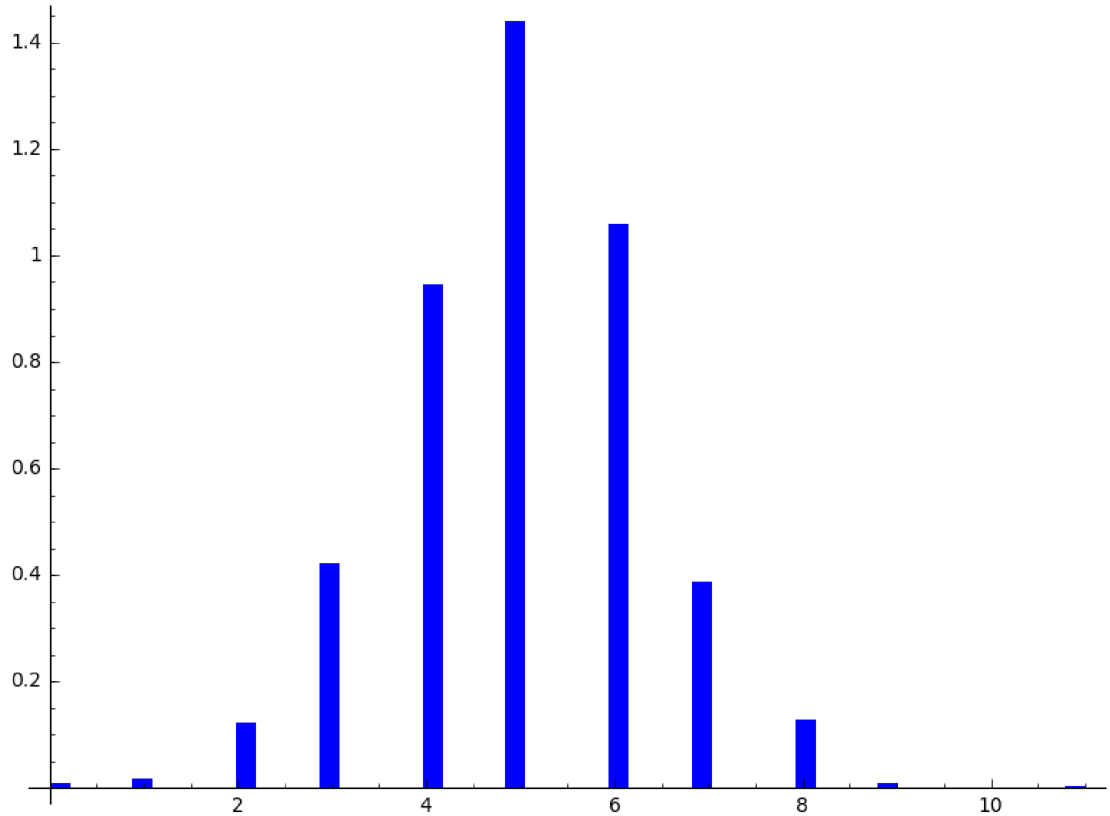
In [83]:
```
T0 = finance.TimeSeries(listaHistogramas[0])
T0.plot_histogram()
```

Out[83]:



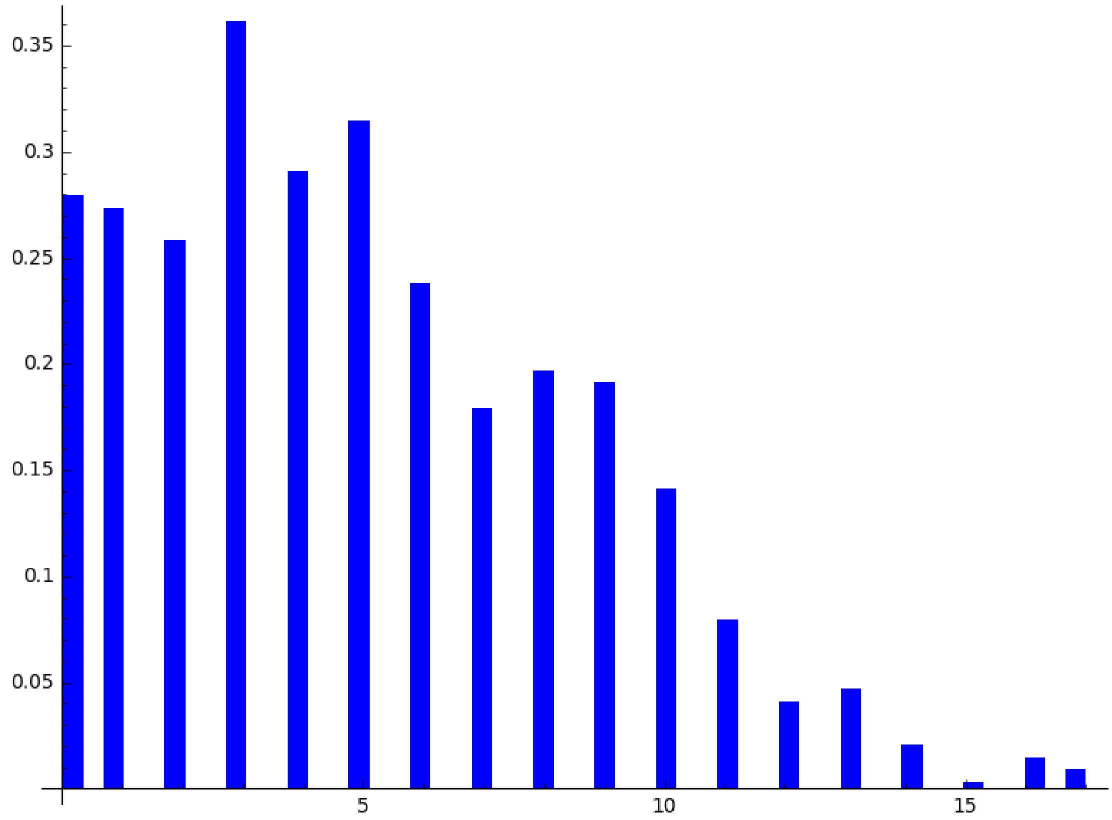

```
In [84]: T1 = finance.TimeSeries(listaHistogramas[1])
         T1.plot_histogram()
```

Out[84]:



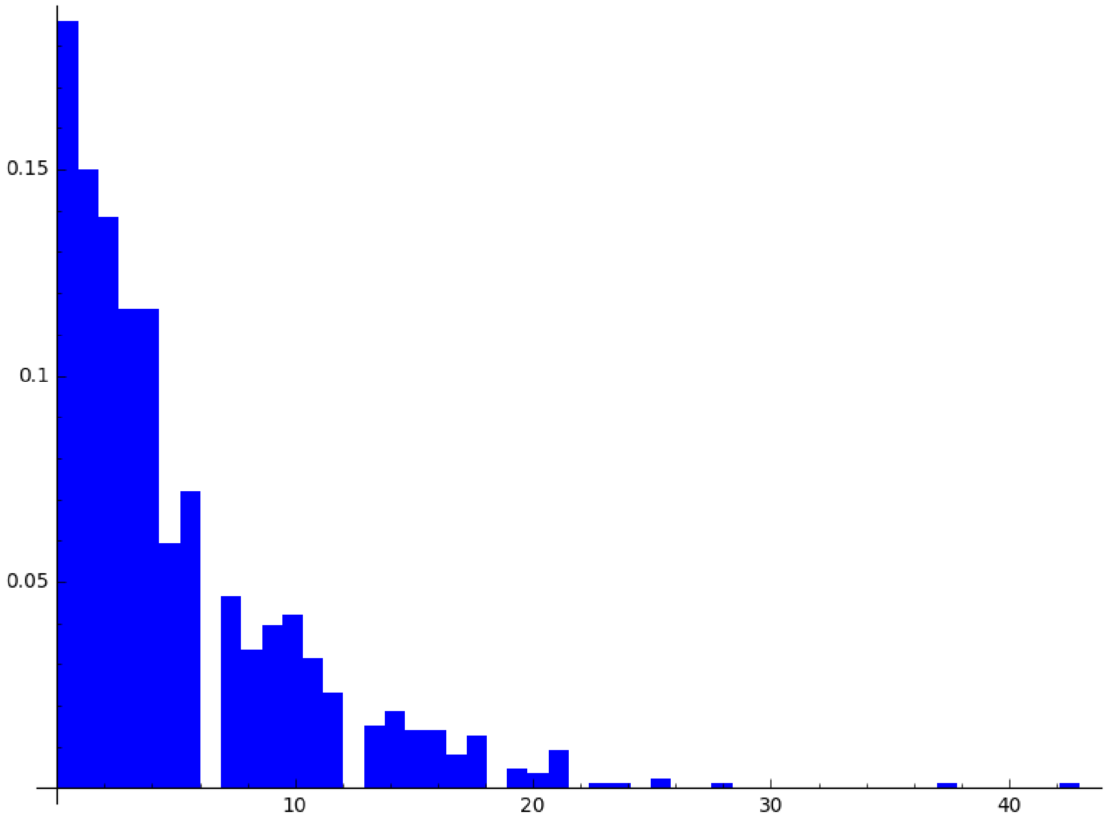

```
In [85]: T2 = finance.TimeSeries(listaHistogramas[2])
         T2.plot_histogram()
```

Out[85]:



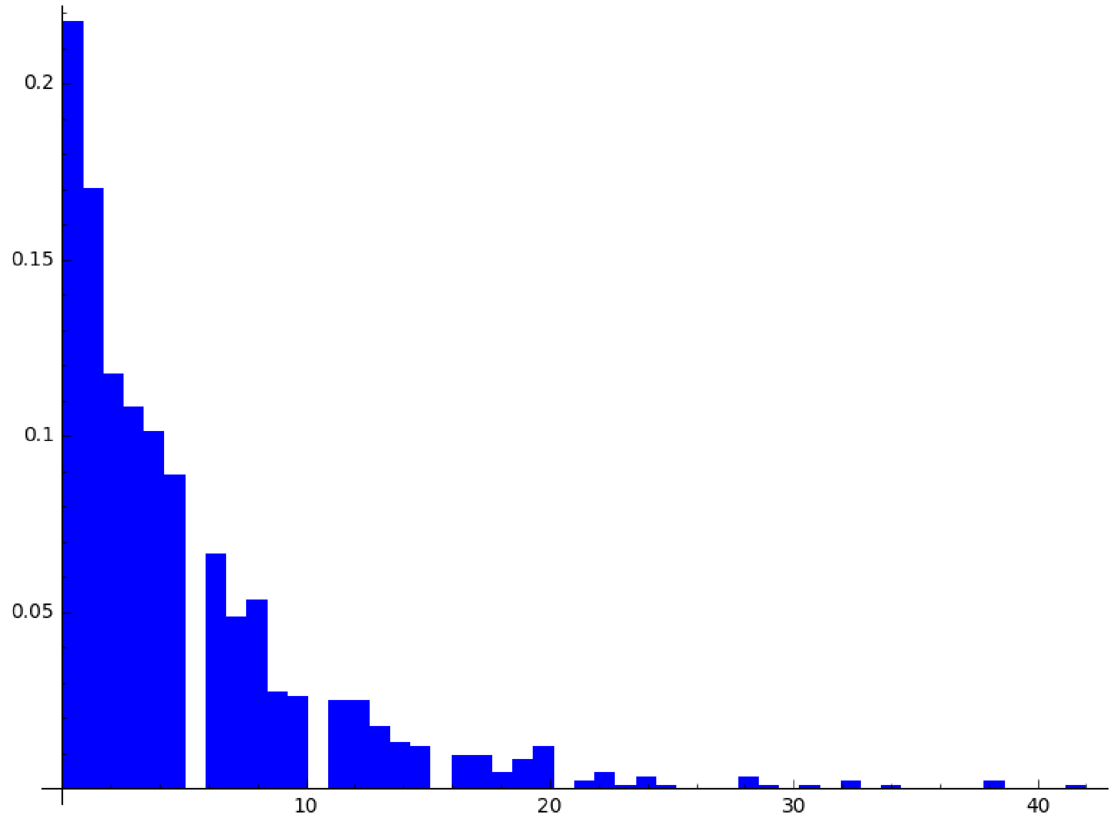

```
In [86]: T3 = finance.TimeSeries(listaHistogramas[3])
         T3.plot_histogram()
```

Out[86]:




```
In [87]: T4 = finance.TimeSeries(listaHistogramas[4])
         T4.plot_histogram()
```

Out[87]:



Pasos 2-bidimensionales
Vamos a calcular la distancia promedio al origen en el instante N.

```
In [92]: def randDir(P):
             x = randint(1,4)
             if x==1:
                 P[0] += 1
             elif x==2:
                 P[0] -= 1
             elif x==3:
                 P[1] += 1
             else:
                 P[1] -= 1
             return P
```

```
In [105]: def randWalkBidim(tiempo):
              P = [0,0]
              for i in xsrange(tiempo):
                  P = randDir(P)
              dist = sqrt((P[0]**2)+(P[1]**2))
              return dist.n(digits=4)
```

```
In [106]: print randWalkBidim(100)
```

18.11

```
In [118]: total = 0;
          for i in xsrange(1000):
              Aux1 = [randWalkBidim(100*k) for k in xsrange(1,10)]
              a = sum(Aux1)/len(Aux1)
              total += a
          print("Media total: "+str(total/1000))
```

```
Media total: 19.05
```

Para un tiempo entre 100 y 1000 obtenemos una distancia media del origen de 19 u.
Vamos ahora intentar calcular la probabilidad de que la persona vuelva al origen con paseos aleatorios bidimensionales.

```
In [119]: def Comeback(tiempo): #tiempo es la unidad de tiempo que le damos para que vuelva al
              P = [0,0]
              for i in xsrange(tiempo):
                  P = randDir(P)
                  if P[0]==0 and P[1]==0:
                      return 1
              return 0
```

Vamos a probarlo con un tiempo de 100 unidades:

```
In [124]: tot = 0
          for i in xsrange(10000):
              tot += Comeback(100)
          prob1 = (tot/10000).n(digits=4)
          print("Probabilidad que vuelva con tiempo 100: "+str(prob1))
```

```
Probabilidad que vuelva con tiempo 100: 0.5808
```

Vamos a probarlo ahora con un tiempo de 1000 unidades:

```
In [125]: tot2 = 0
          for i in xsrange(10000):
              tot2 += Comeback(1000)
          prob2 = (tot2/10000).n(digits=4)
          print("Probabilidad que vuelva con tiempo 100: "+str(prob2))
```

```
Probabilidad que vuelva con tiempo 100: 0.6824
```

Cerca del 70%
Urnas de Polya

```
In [2]: def nuevaUrna(L):
            AUX = copy(L)
            x = randint(0, len(L)-1)
            if L[x] == 0:
                AUX.append(0)
            elif L[x] == 1:
                AUX.append(1)
            else:
                print("Error")
                return -1
            return AUX
```

```
In [3]: L = [1, 0]
        L = nuevaUrna(L)
        print L
```

```
[1, 0, 1]
```

```
In [4]: def conjuntoUrnas(N):
            L = [0,1]
            T = []
            T.append(L)
            for i in xsrange(N):
                L = nuevaUrna(L)
                T.append(L)
            return T,T[-1]
```

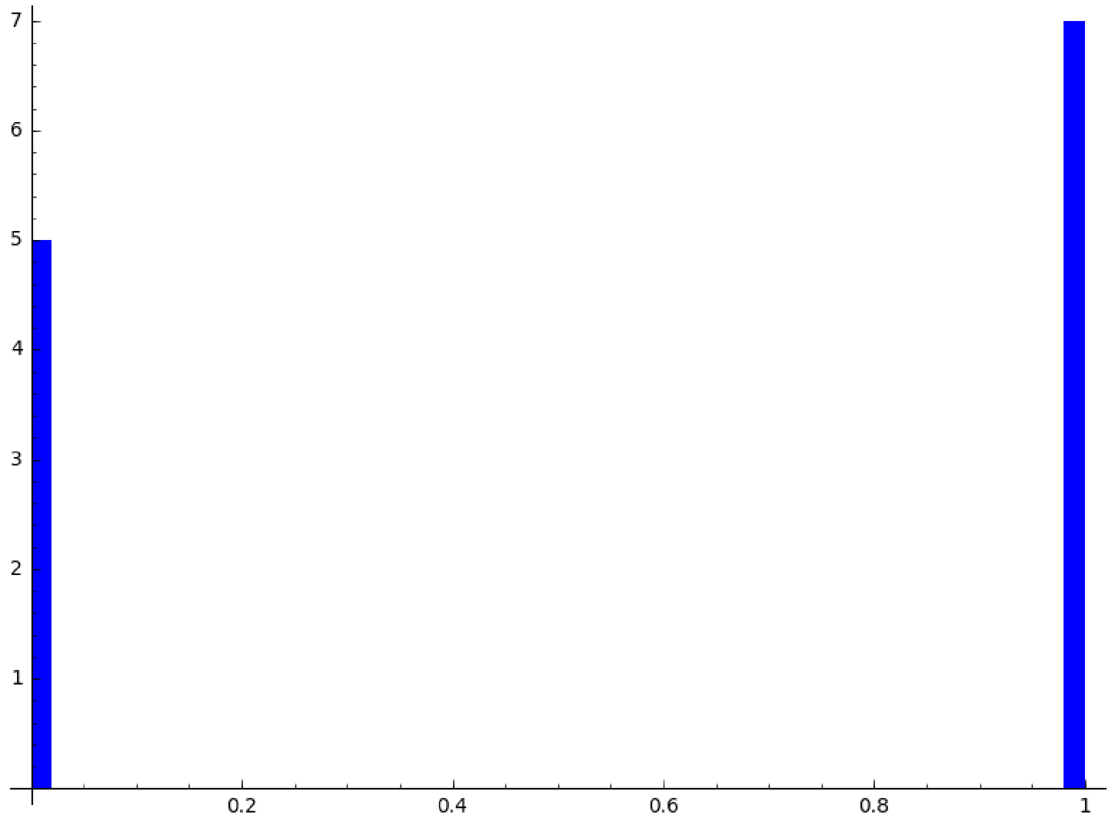
```
In [5]: Q = conjuntoUrnas(10)
        print("Lista de urnas: ")
        print Q[0]
        print ("Ultima urna: ")
        print Q[1]
```

```
Lista de urnas:
[[0, 1], [0, 1, 0], [0, 1, 0, 0], [0, 1, 0, 0, 1], [0, 1, 0, 0, 1, 1], [0, 1, 0, 0, 1, 1, 1],
Ultima urna:
[0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1]
```

```
In [6]: Histo1 = finance.TimeSeries(Q[1])
        Histo1.plot_histogram(normalize=False)
```

```
/usr/local/SageMath/local/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserWarn
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.'
```

```
Out[6]:
```




```
In [11]: def probPolya(L):
             #consideramos los 1's como bolas blancas
             return (sum(L)/len(L)).n(digits=7)

In [12]: LCHECK = [1, 0, 1, 1, 0, 0, 0, 1, 0 , 1, 1]
         print probPolya(LCHECK)

0.5454545

In [13]: def NprobabilidadesPolya(N):
             L = [0, 1]
             T = []
             T.append(probPolya(L))
             for i in xsrange(N):
                 L = nuevaUrna(L)
                 T.append(probPolya(L))
             return T

In [17]: TCHECK = NprobabilidadesPolya(10)
         print TCHECK
```

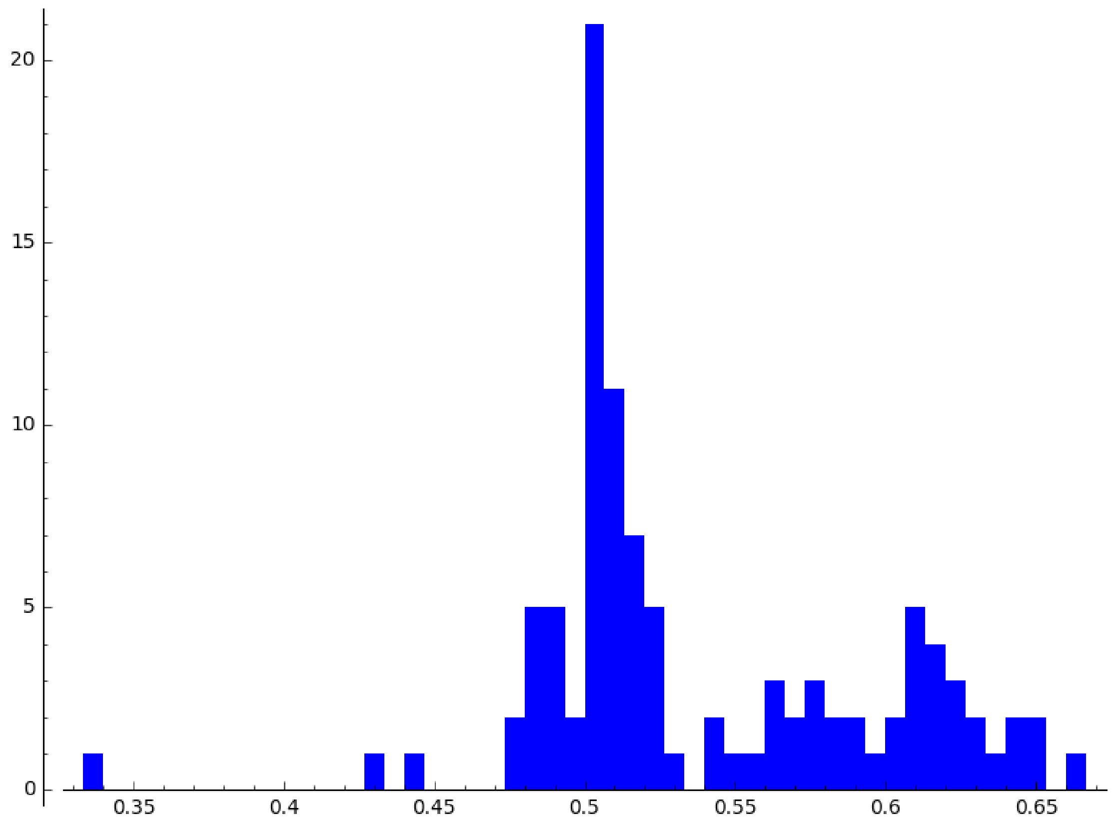
```
[0.5000000, 0.3333333, 0.5000000, 0.4000000, 0.5000000, 0.4285714, 0.3750000, 0.3333333, 0.300
```

In [30]:
```
TCHECK = NprobabilidadesPolya(100)
print TCHECK
```

```
[0.5000000, 0.3333333, 0.5000000, 0.6000000, 0.5000000, 0.4285714, 0.5000000, 0.4444444, 0.500
```

In [31]:
```
Histo2 = finance.TimeSeries(TCHECK)
Histo2.plot_histogram(normalize=False)
```

Out[31]:

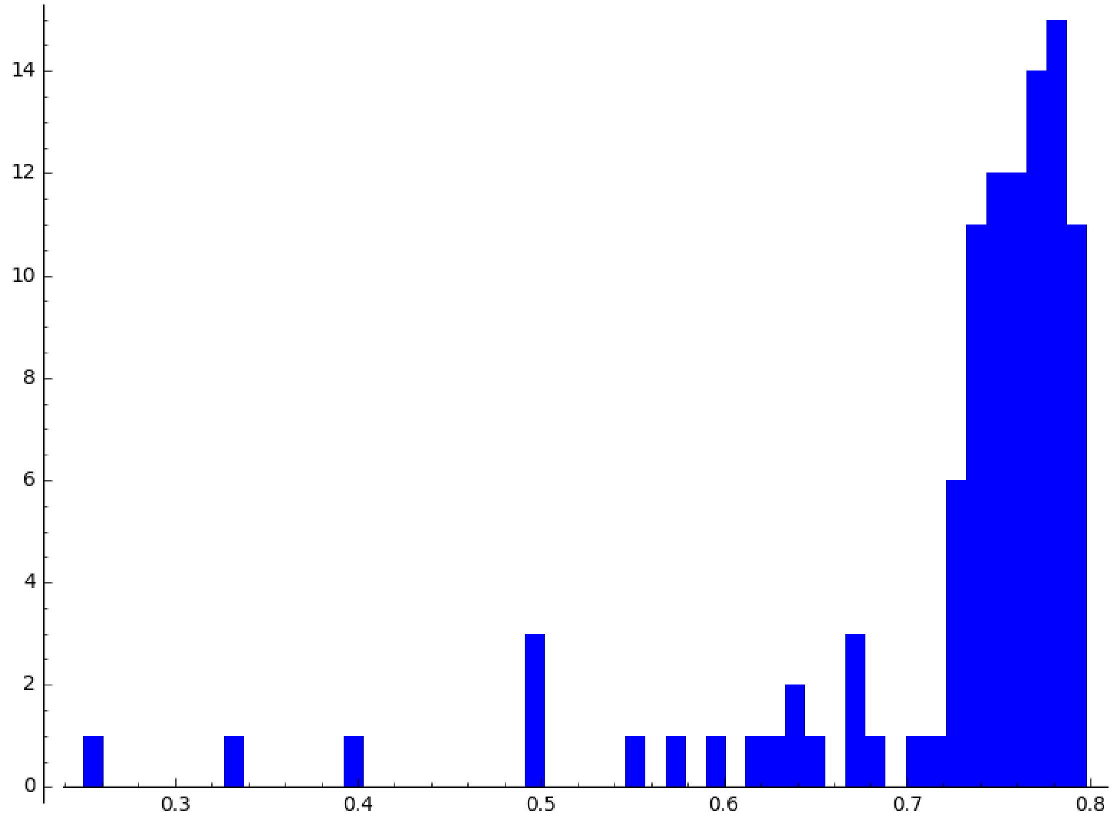


In [28]:
```
TCHECK2 = NprobabilidadesPolya(100)
print TCHECK2
```

```
[0.5000000, 0.3333333, 0.2500000, 0.4000000, 0.5000000, 0.5714286, 0.5000000, 0.5555556, 0.600
```

In [29]:
```
Histo3 = finance.TimeSeries(TCHECK2)
Histo3.plot_histogram(normalize=False)
```

Out[29]:

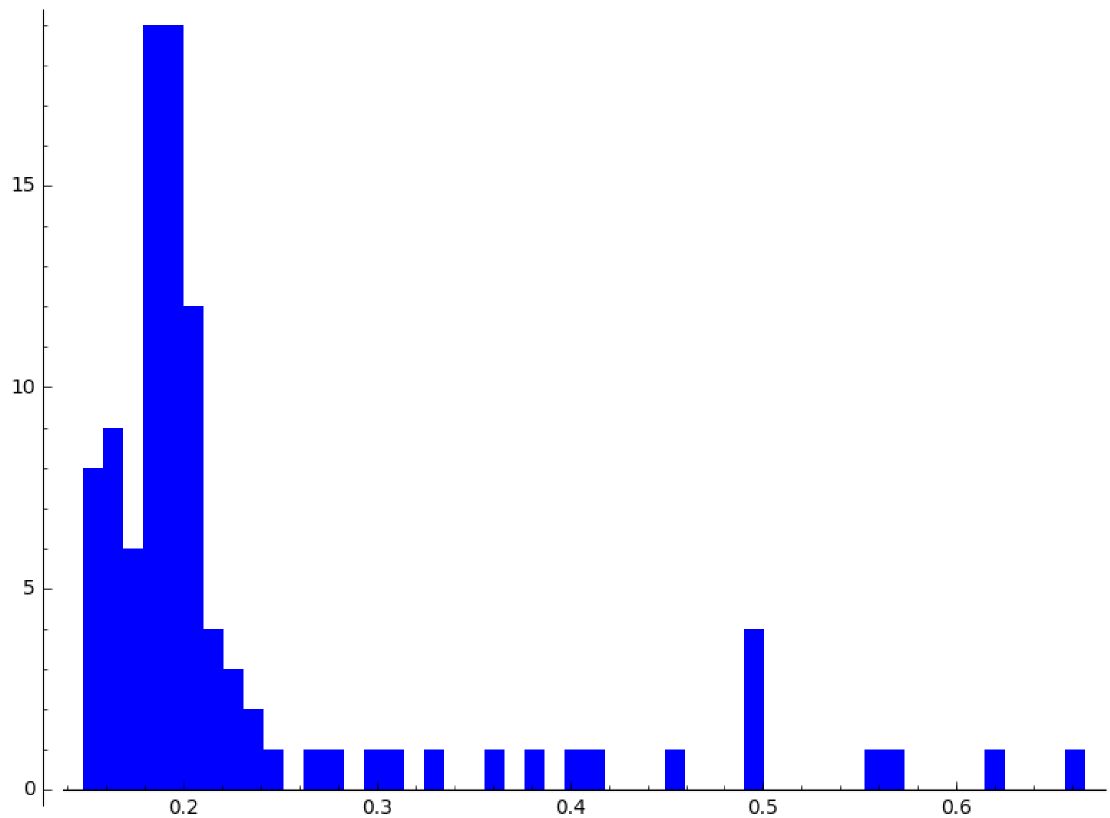


In [36]:
```
TCHECK3 = NprobabilidadesPolya(100)
print TCHECK3
```

```
[0.5000000, 0.6666667, 0.5000000, 0.4000000, 0.5000000, 0.5714286, 0.6250000, 0.5555556, 0.5000
```

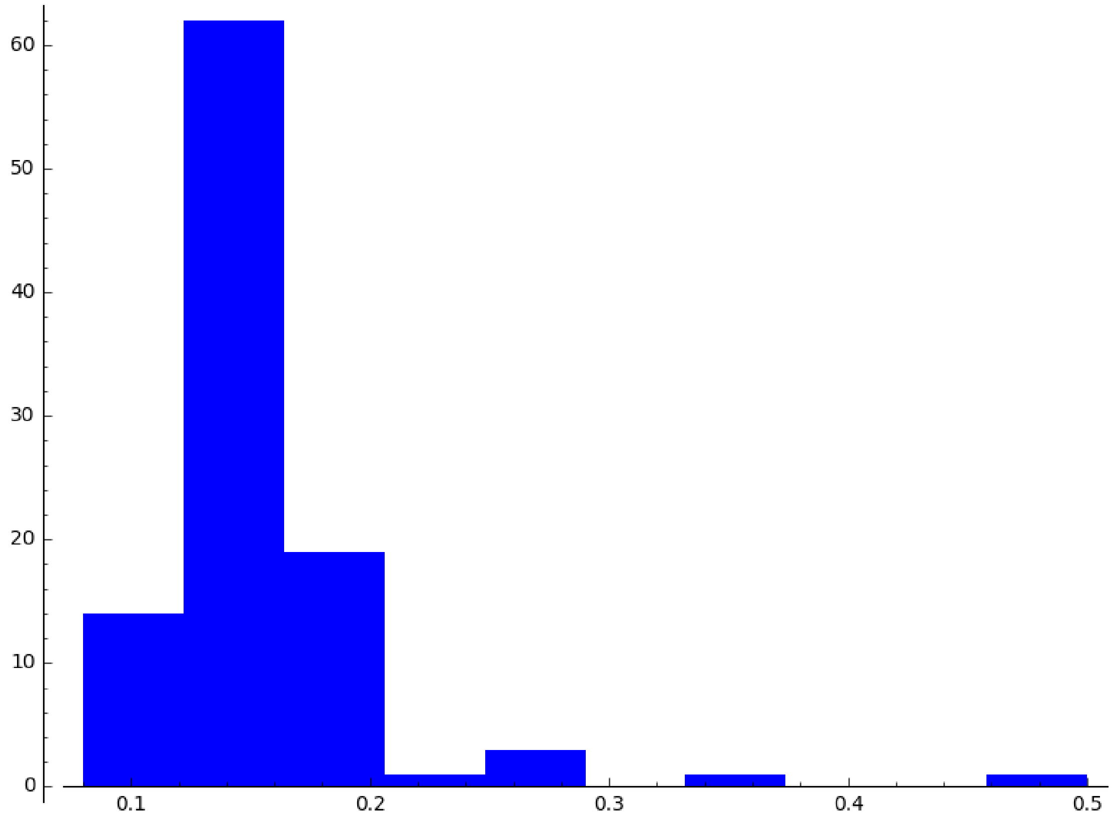
In [37]:
```
Histo4 = finance.TimeSeries(TCHECK3)
Histo4.plot_histogram(normalize=False)
```

Out[37]:



Se puede ver después de algunos ejemplos que las probabilidades P(i) se concentran en un cierto intervalo después de un número considerable de "nuevas urnas". El intervalo de congregación depende del desarrollo de las primeras urnas, ya que dos bolas blancas seguidas tiene un gran impacto en las urnas posteriores ya que tiene mucha más probabilidad otra bola blanca.

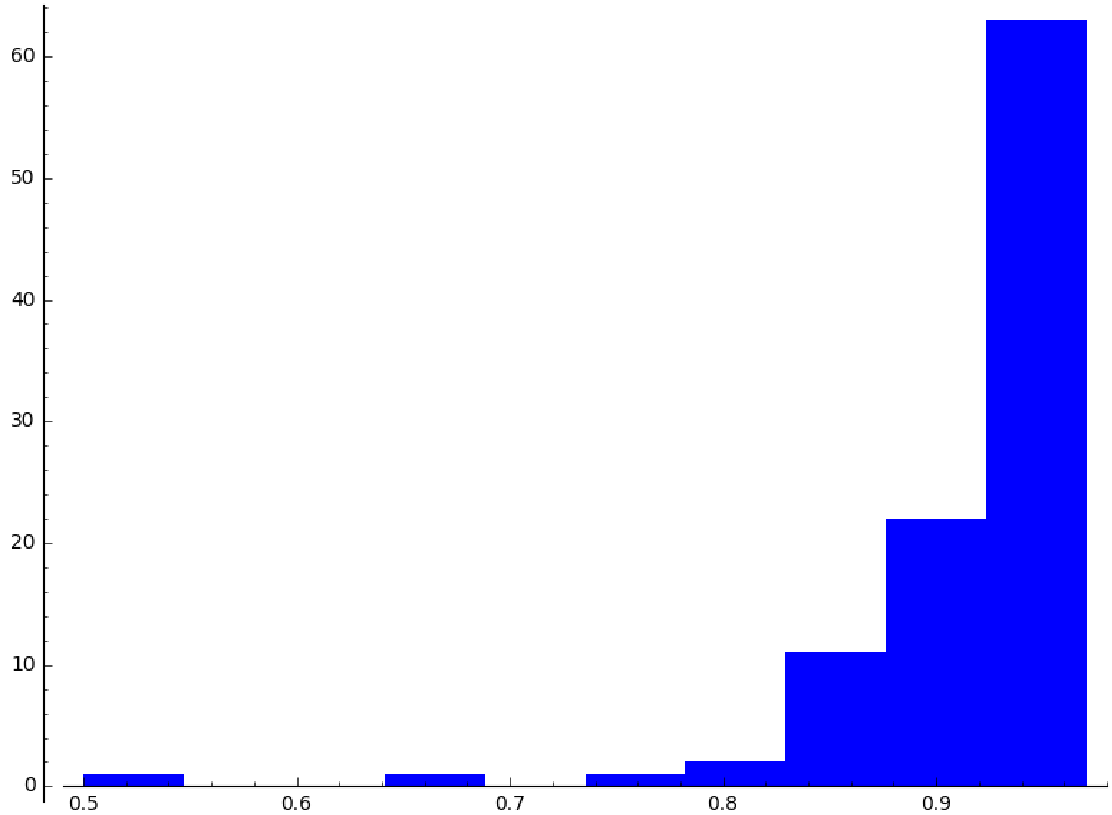
```
In [41]: def comportamientoPolya(N, n):
             T = []
             for i in xsrange(n):
                 Laux = NprobabilidadesPolya(N)
                 Histo = finance.TimeSeries(Laux)
                 T.append(Histo)
             return T
                 #Intentaba que la funcion mostrase los histogramas, pero no los muestra...
                 #Histo.plot_histogram(bins=10, normalize=False)

In [42]: Taux = comportamientoPolya(100, 10)

In [45]: Taux[0].plot_histogram(bins=10, normalize=False)

Out[45]:
```




```
In [46]: #No consigo que se plotee un conjunto de histogramas.
         for i in xsrange(len(Taux)):
             Taux[i].plot_histogram(bins=10, normalize=False)
```

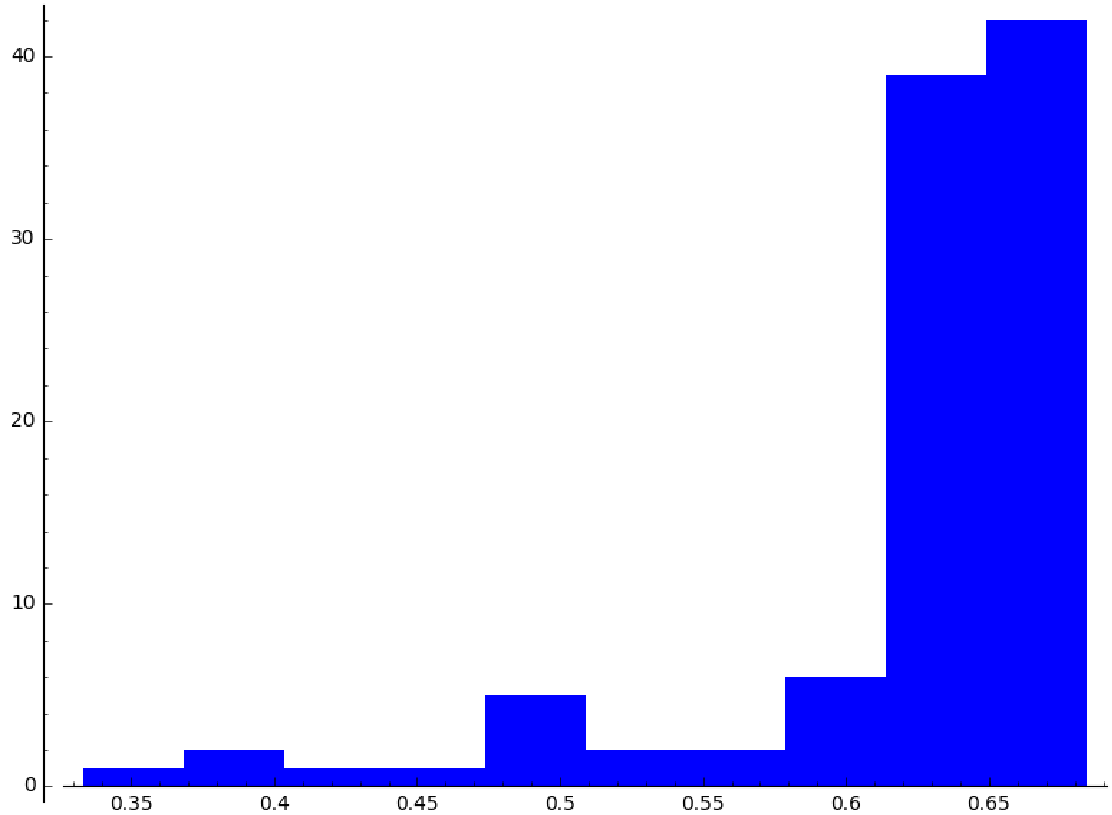
```
In [49]: Taux[4].plot_histogram(bins=10, normalize=False)
```

```
Out[49]:
```



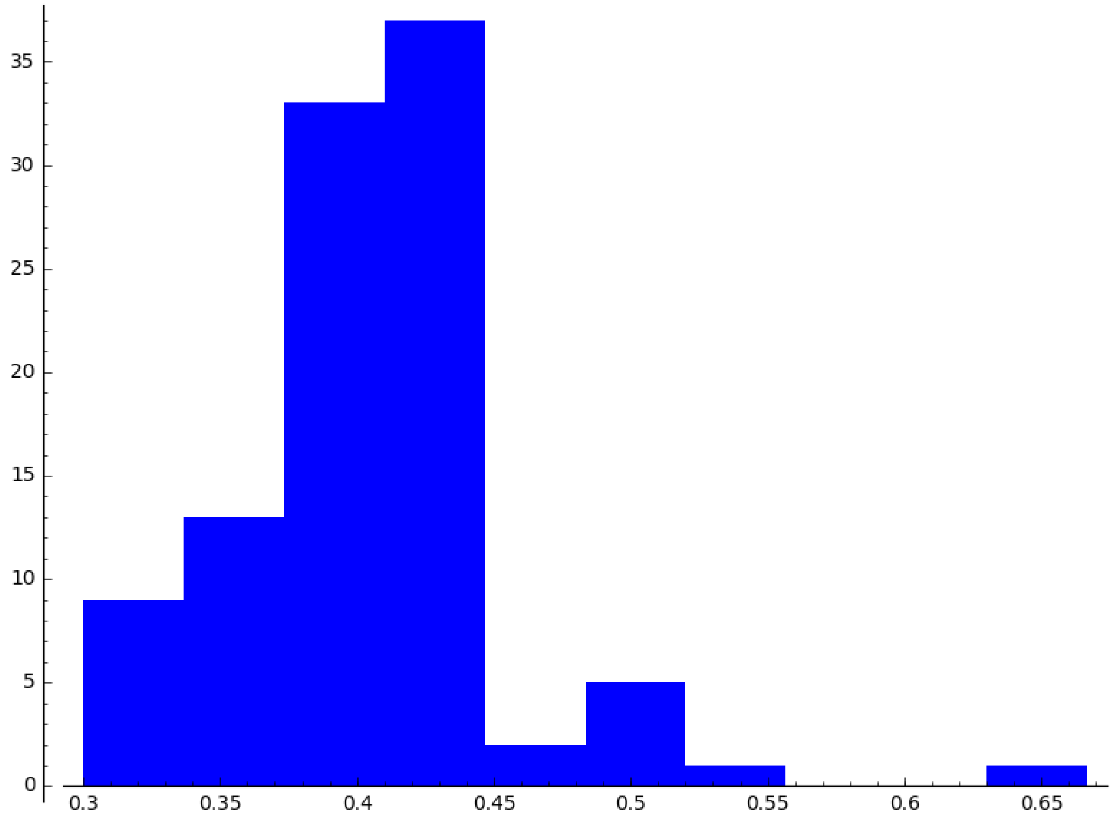

```
In [51]: Taux[6].plot_histogram(bins=10, normalize=False)
```

Out[51]:




```
In [52]: Taux[7].plot_histogram(bins=10, normalize=False)
```

Out[52]:



Estos son algunos ejemplos. Se vuelve apreciar lo comentado anteriormente.