Parcial computacional - Métodos estadísticos en física experimental

María Paula Caral

In [1]:
```
from __future__ import division
import random
import numpy as np
import math
import matplotlib.pyplot as plt
import string
from math import factorial, log
```

2. Estadísitica en la calle

In [2]:
```
# Para asignarle un entero a cada una de las patentes que observé, lo que hago es cre
ar una lista de todas las patentes posibles
# y luego "matcheo" mis patentes con esa lista, guardando el índice que les correspod
e.
# Para poder armar la lista de todas las patentes posibles, defino alfabeto y número
s:
alfabeto =  []

for i in range(26):
    a = string.ascii_uppercase[i]
    alfabeto.append(a)

numeros = [0,1,2,3,4,5,6,7,8,9]
```

In [3]:
```
# Genero un vector con todas las patentes posibles
todas_las_patentes = []

for n in range (4): # Sé que de las más nuevas que vi empezaban con AD, entonces ni m
e gasto en generar el resto
    for m in range(len(numeros)):
        for l in range (len(numeros)):
            for k in range (len(numeros)):
                for j in range (len(alfabeto)):
                    for i in range(len(alfabeto)):
                        p =  ['A',alfabeto[n],numeros[m],numeros[l],numeros[k],alfabe
to[j],alfabeto[i]]
                        todas_las_patentes.append(p)
```

In [4]:
```
# Importo mis patentes

mis_patentes = [['A','B',9,3,8,'E','O'],['A','B',3,8,8,'J','D'],['A','A',0,6,2,'X',
'T'],['A','A',5,4,1,'L','N'],['A','C',1,3,4,'M','C'],['A','B',4,1,6,'L','Q'],['A','A'
,6,8,3,'Q','B'],['A','C',3,5,5,'H','V'],['A','D',4,6,2,'S','H'],['A','A',7,7,0,'A',
'D'],['A','B',9,1,5,'L','X'],['A','C',4,8,9,'F','V'],['A','B',8,9,5,'N','M'],['A','C'
,7,9,3,'B','D'],['A','B',1,9,8,'X','N'],['A','A',3,6,0,'J','E'],['A','D',4,0,6,'W',
'L'],['A','B',6,7,0,'Y','Y'],['A','A',4,7,4,'G','E'],['A','C',3,2,1,'N','Z'],['A','B'
,1,5,0,'H','I'],['A','B',3,7,3,'Z','S'],['A','D',1,9,8,'H','L'],['A','C',8,2,0,'P',
'E'],['A','B',9,3,9,'S','L'],['A','C',7,2,5,'K','A'],['A','D',1,8,8,'Y','K'],['A','C'
,3,5,4,'U','V'],['A','B',6,2,9,'I','B'],['A','B',8,8,0,'S','U'],['A','A',2,0,6,'O',
'T'],['A','A',0,3,3,'U','Z'],['A','D',1,6,2,'H','J'],['A','C',2,5,2,'D','X'],['A','A'
,8,9,7,'U','N'],['A','C',9,6,6,'A','D'],['A','C',9,5,3,'B','X'],['A','C',1,8,7,'R',
'J'],['A','C',4,5,1,'D','A'],['A','A',1,4,5,'W','R'],['A','A',9,7,3,'F','A'],['A','B'
,3,6,2,'T','C'],['A','D',2,1,0,'U','C'],['A','C',3,5,4,'U','L'],['A','D',1,2,5,'K',
'C'],['A','B',5,9,9,'O','B'],['A','D',2,2,0,'O','W'],['A','A',5,2,3,'C','M'],['A','A'
,0,8,1,'V','P'],['A','B',2,3,8,'Z','P'],['A','A',1,4,0,'W','F'],['A','C',2,4,7,'B',
'I'],['A','C',2,7,2,'X','G'],['A','B',3,0,8,'J','P'],['A','A',8,4,1,'Q','U'],['A','C'
,9,7,3,'K','G'],['A','A',4,9,8,'X','I'],['A','C',6,9,4,'A','M'],['A','B',7,7,1,'E',
'K'],['A','B',3,7,9,'W','R'],['A','B',6,7,7,'F','Y'],['A','C',5,8,9,'F','J'],['A','D'
,0,6,8,'S','C'],['A','A',9,2,8,'K','B'],['A','B',2,8,1,'X','G'],['A','D',2,9,3,'D',
'V'],['A','B',6,8,1,'X','D'],['A','B',7,7,4,'X','V'],['A','D',1,2,8,'C','A'],['A','B'
,8,0,4,'X','Y'],['A','A',6,9,9,'F','G'],['A','B',6,7,5,'V','A'],['A','B',4,2,7,'J',
'V'],['A','C',4,5,7,'P','W'],['A','B',1,0,1,'D','M'],['A','C',5,8,3,'E','C'],['A','B'
,6,6,3,'P','K'],['A','A',6,3,7,'X','A'],['A','C',2,0,6,'W','T'],['A','A',5,9,5,'N',
'H'],['A','C',5,4,9,'J','H'],['A','B',0,9,6,'D','W'],['A','B',3,3,3,'H','E'],['A','A'
,0,5,1,'S','M'],['A','D',2,9,9,'J','L'],['A','D',4,5,1,'S','D'],['A','D',0,9,9,'T',
'C'],['A','C',6,6,0,'R','P'],['A','B',4,9,2,'Y','D'],['A','C',9,3,1,'F','C'],['A','A'
,8,8,9,'V','P'],['A','A',6,4,9,'U','J'],['A','B',2,8,3,'T','Q'],['A','B',6,4,8,'X',
'D'],['A','A',8,4,1,'Q','X'],['A','A',7,5,1,'A','X'],['A','D',3,9,5,'K','Z'],['A','D'
,0,4,4,'W','Z'],['A','D',1,2,0,'K','L'],['A','D',3,0,9,'K','Y'],['A','A',4,5,0,'H',
'F'],['A','B',7,3,1,'Y','U'],['A','A',5,1,1,'W','C'],['A','C',1,6,5,'F','Q'],['A','A'
,8,3,1,'B','V'],['A','B',2,8,9,'O','D'],['A','A',8,2,7,'D','Z'],['A','D',1,4,1,'A',
'W'],['A','A',1,0,4,'H','S'],['A','A',8,4,1,'Q','Y'],['A','B',4,6,2,'A','Y'],['A','A'
,7,4,6,'K','O'],['A','A',0,6,6,'V','Q'],['A','C',1,4,9,'Q','C'],['A','A',6,5,7,'T',
'N'],['A','A',5,1,3,'D','A'],['A','B',5,2,6,'S','Z'],['A','A',0,4,7,'M','I'],['A','A'
,6,2,7,'V','Z'],['A','A',8,2,7,'A','X'],['A','A',7,7,6,'B','K'],['A','D',1,2,0,'J',
'O'],['A','A',6,0,2,'V','A'],['A','A',6,6,2,'L','B'],['A','D',0,7,1,'G','X'],['A','B'
,6,5,3,'X','W'],['A','B',9,1,5,'H','S'],['A','A',4,2,9,'K','V'],['A','A',1,8,2,'L',
'M'],['A','C',8,2,9,'O','Y'],['A','B',1,3,9,'E','Z'],['A','A',6,9,9,'T','T'],['A','B'
,9,6,6,'C','O'],['A','A',5,2,2,'T','L'],['A','B',5,2,6,'T','E'],['A','A',0,8,2,'J',
'V'],['A','D',2,2,8,'L','U'],['A','C',6,7,0,'M','E'],['A','B',3,4,9,'J','D'],['A','A'
,4,0,5,'N','H'],['A','B',1,7,9,'O','V'],['A','D',4,8,0,'Y','V'],['A','D',1,7,9,'Q',
'Q'],['A','B',1,4,6,'C','B'],['A','C',9,1,5,'Q','Z'],['A','B',7,5,1,'Y','L'],['A','A'
,7,9,7,'A','O'],['A','D',4,1,7,'B','D'],['A','A',7,1,5,'F','U'],['A','C',8,9,6,'E',
'O'],['A','B',6,5,2,'Z','Q'],['A','C',8,9,1,'Y','D'],['A','B',1,2,6,'F','O'],['A','C'
,4,5,9,'A','K'],['A','A',1,0,9,'A','H']]
```

In [5]:
```
# "Matcheo" y guardo los índices

lista_de_enteros = [] # Vector donde voy a guardar los enteros que le correspondan a
 cada patente que anoté

for j in range(len(mis_patentes)):
    if mis_patentes[j] in todas_las_patentes:
        i = todas_las_patentes.index(mis_patentes[j])
        lista_de_enteros.append(i) # Guardo todos los índices

#print(lista_de_enteros)
```

In [6]:
```
print('La patente más nueva que vi es', todas_las_patentes[max(lista_de_enteros)])

# Estos valores los voy a usar a lo largo del parcial, así que ya los defino.
a = min(lista_de_enteros)
b = max(lista_de_enteros)
k = len(lista_de_enteros)
```

```
La patente más nueva que vi es ['A', 'D', 4, 8, 0, 'Y', 'V']
```

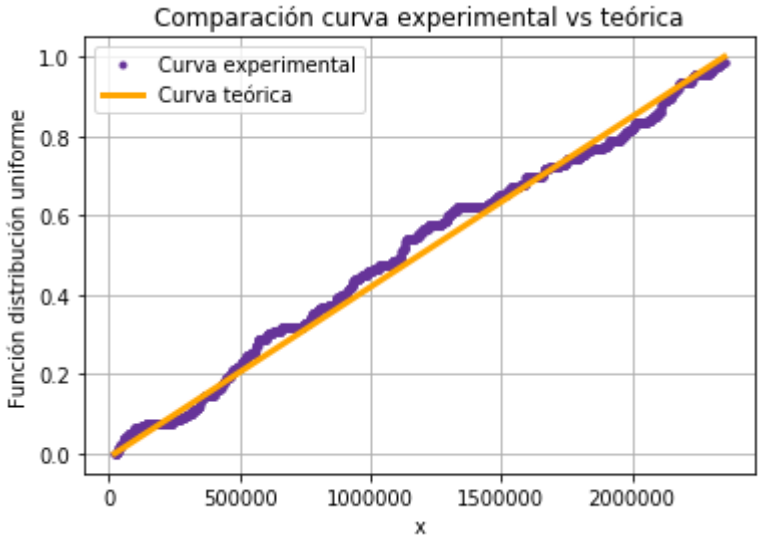
3. ¿Uniformemente distribuidas?

3.1 Test de Kolmogorov-Smirnov

Mi hipótesis nula será "H_0: Mis datos provienen de una distribución uniforme." y voy a testearla usando el estadístico propuesto por el test de K-S.

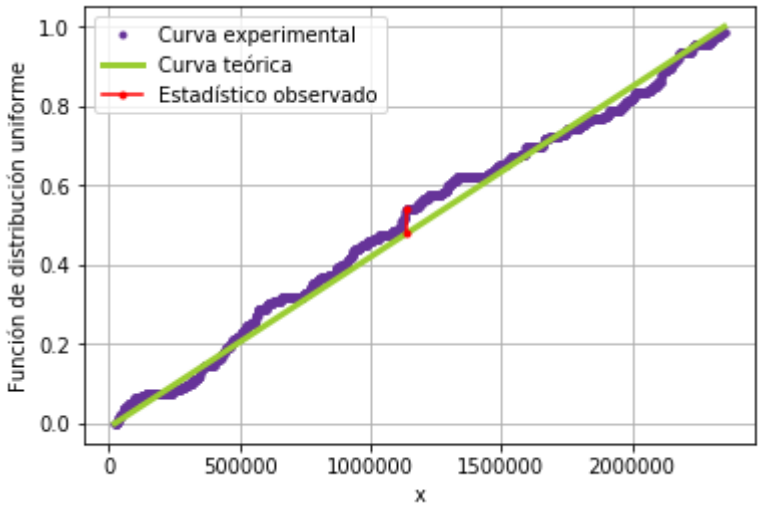
In [7]:
```
# Para hacer el test de K-S necesito ordenar la lista de enteros de mis patentes de m
enor a mayor.
lde_ordenadas = sorted(lista_de_enteros)

# Genero un vector "continuo" entre los enteros que tengo, que me va a ser útil para
 los cálculos y para el gráfico.
x = np.arange(a,b,1000)
```

In [8]:
```
# Calculo la curva experimental según el test de K-S

S = [] # Vector donde guardo mi curva experimental

for i in range (len(lde_ordenadas)):
    for j in range (len(x)):
        if lde_ordenadas[i] <= x[j] <= lde_ordenadas[i+1]:
            s = i/k
            S.append(s)


# Calculo la curva teórica correspondiente a la distribución uniforme
c = b-a
Y = []

for i in range(len(x)):
    y = (x[i]-a)/c
    Y.append(y)


plt.plot(x,S,'.',label='Curva experimental',color='rebeccapurple'); # Grafico curva experimental
plt.plot(x,Y,label='Curva teórica',color='orange',linewidth=3) # Grafico curva teórica
plt.xlabel('x');
plt.ylabel('Función distribución uniforme')
plt.legend(loc='upper left');
plt.grid(True)
plt.title('Comparación curva experimental vs teórica');
```



In [9]:
```
# Calculo el estadístico de K-S para mi conjunto de datos
t = []
for j in range(len(S)):
    T = abs(S[j]-Y[j])
    t.append(T)

t_estadistico_medido = max(t)
print('El estadístico t medido es', t_estadistico_medido)

# Lo voy a agregar al gráfico anterior
I = t.index(t_estadistico_medido)

x1, y1 = [x[I], x[I]], [S[I], Y[I]]

plt.plot(x,S,'.',label='Curva experimental',color='rebeccapurple'); # Grafico curva experimental
plt.plot(x,Y,label='Curva teórica',color='yellowgreen',linewidth=3) # Grafico curva teórica
plt.plot(x1,y1,'r.-',label='Estadístico observado') # Grafico el estadístico
plt.xlabel('x');
plt.ylabel('Función de distribución uniforme')
plt.legend(loc='upper left');
plt.grid(True)
```

```
El estadístico t medido es 0.06345056880525873
```



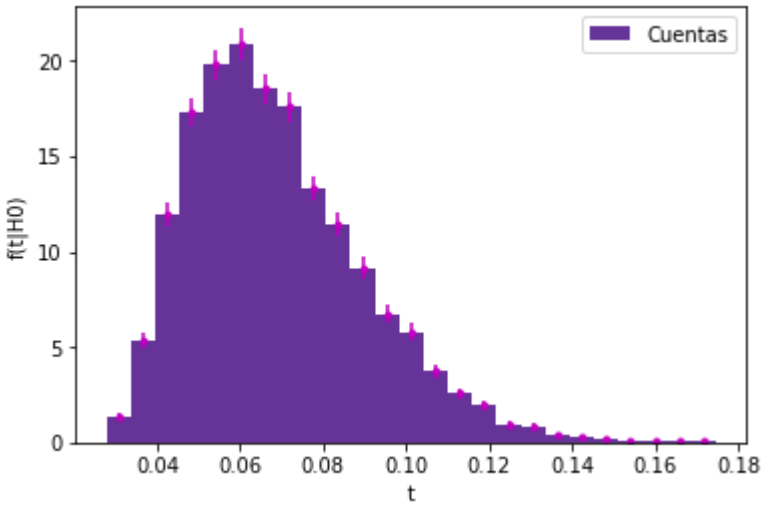
Con el test de Kolmogorov-Smirnov, puedo conocer el percentil que le corresponde a una significancia de $\alpha = 0.05$ utilizando la fórmula que corresponda a mi cantidad de datos. En particular, para esa significancia tengo que $a = \frac{1.36}{\sqrt{n}} = \frac{1.36}{\sqrt{151}} = 0.1107$ (ya que tengo 151 datos). Entonces, siendo el valor de mi t estadístico observado menor que a, puedo **aceptar la hipótesis nula de que los datos provienen de una distribución uniforme**. Esta comparación se ve graficamente en las figuras que siguen.

In [10]:

```
# Ahora voy a calcular el p-valor. Para esto, necesito la densidad de probabilidad de
l estadístico t, así puedo calcular la
# integral desde (o hasta) mi estadístico para el p-valor.

# Para hallar la densidad de probabilidad de mis enteros, simulo conjuntos de datos a
sumiento verdadera la hipótesis nula de
# que mis datos provienen de una distribución uniforme, y les calculo su t correspond
iente.


# Simulo 5000 conjuntos de datos. Para esta cantidad tarda aprox 20 min.
t_est = [] # Vector donde voy a guardar los t's que salgan de cada simulación
for n in range(5000):
    R = []
    while len(R)<=k:
        r = random.randint(a,b+1) # Le pido enteros random en el rango de mis patente
s (distribución uniforme)
        if r in R:  # Pido esto antes de appendear en mi vector de mediciones para as
egurarme de no repetir enteros
            R = R
        else:
            R.append(r) # Y ahora que tengo mi conjunto de datos, sigo los mismos pas
os que antes para calcular el estadístico

    R_ordenadas = sorted(R) # Ordeno los enteros de menor a mayor

    S_simul = [] # Vector donde guardo mi curva experimental

    for i in range (len(R_ordenadas)-1):
        for j in range (len(x)):
            if R_ordenadas[i] <= x[j] <= R_ordenadas[i+1]:
                s = i/k
                S_simul.append(s)

    t_simul = []
    for j in range(len(S_simul)-1):
        T = abs(S_simul[j]-Y[j])
        t_simul.append(T)

    t_est.append(max(t_simul))
```

In [11]:
```
# Hago el histograma de los t_est simulados y obtengo la densidad de probabilidad de
 t dado que H_0 es verdadera.

bines = 25
entradas, bins, patches = plt.hist(t_est,bins=bines,density=True,label='Cuentas',colo
r='rebeccapurple');
#Para los errores del histograma
entradas_ = np.histogram(t_est,bins=bines)[0]
bin_centers = 0.5 * (bins[:-1] + bins[1:])
bin_width = bins[1] - bins[0]
yerr=np.sqrt(entradas_)/(bin_width*sum(entradas_))

plt.errorbar(bin_centers, entradas, yerr, fmt='m.')
plt.xlabel('t');
plt.ylabel('f(t|H0)')
plt.legend(loc='upper right');
print('La distribución del estadístico t dado que H0 es cierta está dada por')
```

```
La distribución del estadístico t dado que H0 es cierta está dada por
```

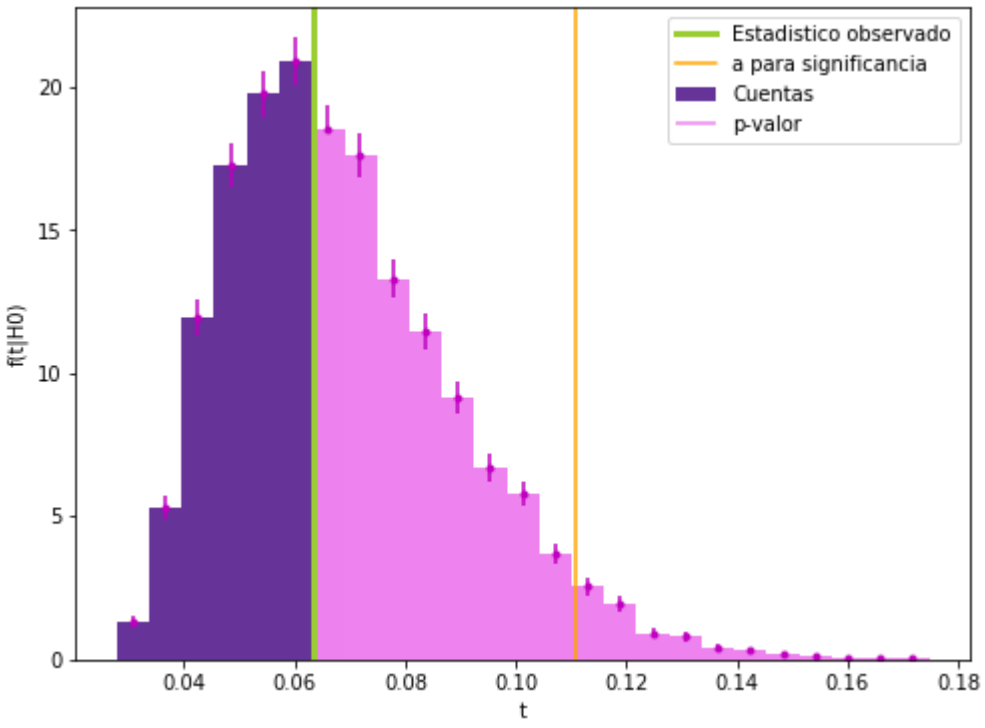


In [12]:
```
# Ahora que tengo la distribución puedo calcular el p-valor.
# Para eso, voy a sumar el área de los bins desde el que corresponde a t_estadistico_
medido (el de mis mediciones) hasta el
# último.


# Entonces busco el índice del bin que le corresponde al t de mis mediciones

i_medido = np.digitize(t_estadistico_medido,bins)

# Para la suma de las areas de los bines calculo el ancho de los mismos

bin_width = bins[1] - bins[0] # ancho de todos los bins

ancho = bins[i_medido]-t_estadistico_medido # ancho de la porcion de bin que tengo qu
e integrar desde el estadistico hasta que
# termina ese bin (si tomara ese bin entero estaría contando probabilidad que no corr
esponde).

# Multiplicando por la altura correspondiente en cada caso y sumando, calculo el p-va
lor
pvalue = bin_width * sum(entradas[i_medido+1:bines]) + entradas[i_medido]*ancho

print('El p-valor para el t medido es', pvalue)
```

```
El p-valor para el t medido es 0.4342024878038623
```

Dado este p-valor y la significancia, se puede ver claramente que puedo aceptar mi hipótesis nula.

In [13]:
```
# Vuelvo a hacer el gráfico de la densidad de t, marcando el estadistico y el pvalor.
entradas, bins, patches = plt.hist(t_est,bins=bines,density=True,label='Cuentas',colo
r='rebeccapurple');

plt.errorbar(bin_centers, entradas, yerr, fmt='m.')

# Sombreo en otro color todo lo que es p-valor
for i in range(0,i_medido):
    patches[i].set_facecolor('rebeccapurple')
for i in range(i_medido,bines):
    patches[i].set_facecolor('violet')

v = np.arange(t_estadistico_medido, bins[i_medido],0.0001)
plt.vlines(v[0],ymin=0,ymax=entradas[i_medido-1],color='violet',label='p-valor')
for j in range(len(v)):
    plt.vlines(v[j],ymin=0,ymax=entradas[i_medido-1],color='violet')

# Marco el estadístico observado en verde
plt.axvline(t_estadistico_medido,color='yellowgreen',linewidth=3,label='Estadistico o
bservado');

# Marco donde esta el a que da una significancia de 0.05
alpha = 0.1107
plt.axvline(alpha,color='orange',label='a para significancia');


plt.xlabel('t');
plt.ylabel('f(t|H0)')
plt.legend(loc='upper right');
fig = plt.gcf()
fig.set_size_inches(8,6)
```



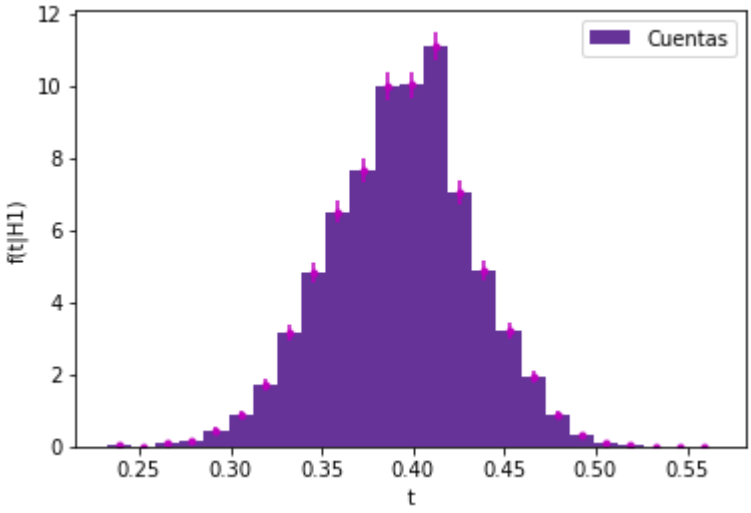
3.2 Hipótesis alternativa: distribución exponencial

Para calcular el poder del test dada esta hipótesis alternativa, tengo que calcular la densidad del estadístico t dado ahora que "H_1: Mis datos provienen de una distribución exponencial" es verdadera. Para eso, hago el mismo procedimiento que antes: simulo conjuntos de datos y a cada uno de ellos les calculo el estadístico correspondiente. Lo que cambia ahora, es que en vez de tomar muestras de una distribución uniforme (randint), uso la distribución exponencial con el parámetro dado (random.exponential).

In [14]:
```
# De nuevo, simulo 5000 conjuntos de datos, pero ahora suponiendo que H1 es verdader
a.
# Con 5000 iteraciones tarda aprox 51 minutos

t_est_alt = []
lam = 4*10**(-7)
beta = 1/lam
for n in range(5000):
    R_alt = []
    while len(R_alt)<=k:
        r_alt = np.random.exponential(scale=beta) # Le pido enteros random en el rang
o de mis patentes (distribución exponencial)
        if r_alt in R_alt:
            R_alt = R_alt
        else:
            R_alt.append(r_alt)

    R_alt_ord = sorted(R_alt) # Ordeno los enteros de menor a mayor

    S_simul_alt = [] # Vector donde guardo mi curva experimental

    for i in range (len(R_alt_ord)):
        for j in range (len(x)):
            if R_alt_ord[i] <= x[j] <= R_alt_ord[i+1]:
                s_alt = i/k
                S_simul_alt.append(s_alt)

    t_simul_alt = []
    for j in range(len(S_simul_alt)-1):
        T_alt = abs(S_simul_alt[j]-Y[j])
        t_simul_alt.append(T_alt)

    t_est_alt.append(max(t_simul_alt))
```

In [15]:
```
# Hago el histograma de los t_est simulados y obtengo la densidad de probabilidad de
 t dado que H_1 es verdadera.

bines_alt = 25
entradas_alt, bins_alt, patches_alt = plt.hist(t_est_alt,bins=bines_alt,density=True,
label='Cuentas',color='rebeccapurple');
#Para los errores del histograma
entradas__alt = np.histogram(t_est_alt,bins=bines_alt)[0]
bin_centers_alt = 0.5 * (bins_alt[:-1] + bins_alt[1:])
bin_width_alt = bins_alt[1] - bins_alt[0]
yerr_alt=np.sqrt(entradas__alt)/(bin_width_alt*sum(entradas__alt))

plt.errorbar(bin_centers_alt, entradas_alt, yerr_alt, fmt='m.')
plt.xlabel('t');
plt.ylabel('f(t|H1)')
plt.legend(loc='upper right');
print('La distribución del estadístico t dado que H1 es cierta está dada por')
```

```
La distribución del estadístico t dado que H1 es cierta está dada por
```

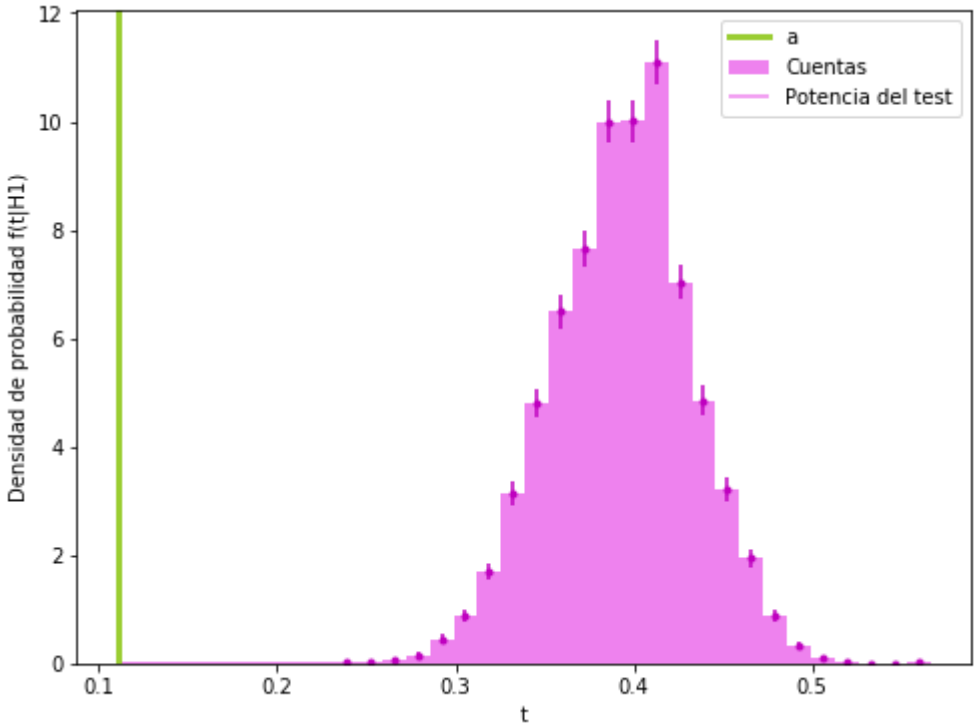


In [16]:
```
# Ahora que tengo la densidad de t|H1, para calcular la potencia del test solo tengo
 que integrar desde el valor de a que
# calculé para la correspondiente significancia e integrar desde ahí hasta el final s
obre esta distribución.

# Entonces busco el índice del bin que le corresponde al t = a (que llamé alpha..)

i_alpha_alt = np.digitize(alpha,bins_alt)

#En este caso, no tengo que contar ningún término especial aparte (tienen probabilida
d despreciable)

# Multiplicando por la altura correspondiente en cada caso y sumando, calculo el p-va
lor

potencia = bin_width_alt * sum(entradas_alt[i_alpha_alt:bines_alt])

print('La potencia del test dada la significancia alpha es', potencia)
```

```
La potencia del test dada la significancia alpha es 1.0
```

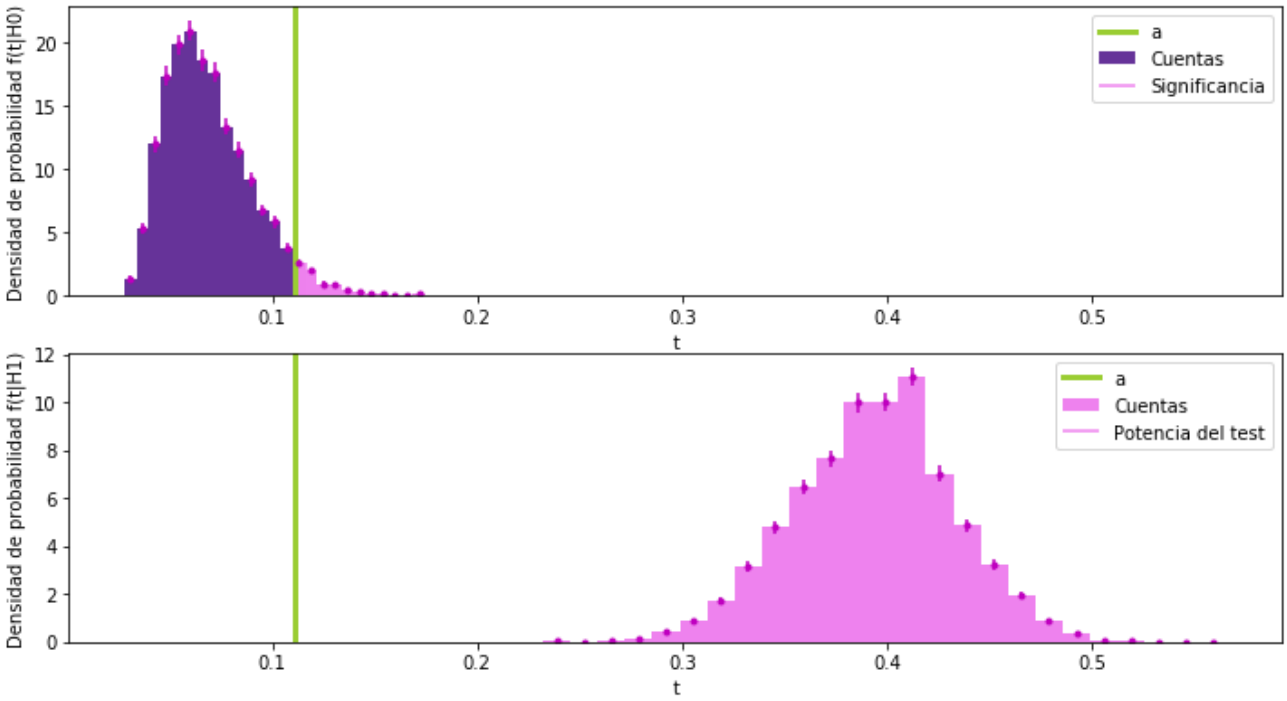
Es decir, el test con esta hipótesis nula es extremadamente poderoso. Al estar bien "desacopladas" las distribuciones del estadístico cuando es verdadera H0 y cuando es verdadera H1, disminuye la probabilidad de error tipo 2 y el test se vuelve muy potente.

In [17]:
```
# Vuelvo a hacer el gráfico de la densidad de t|H1, marcando el "a" y la potencia
entradas_alt, bins_alt, patches_alt = plt.hist(t_est_alt,bins=bines_alt,density=True,
label='Cuentas',color='rebeccapurple');
plt.errorbar(bin_centers_alt, entradas_alt, yerr_alt, fmt='m.')

# Sombreo todo lo que es potencia
for i in range(0,i_alpha_alt):
    patches_alt[i].set_facecolor('rebeccapurple')
for i in range(i_alpha_alt,bines_alt):
    patches_alt[i].set_facecolor('violet')

v_alt = np.arange(alpha, bins_alt[i_alpha_alt],0.0001)
plt.vlines(v_alt[0],ymin=0,ymax=entradas_alt[i_alpha_alt-1],color='violet',label='Pot
encia del test')
for j in range(len(v_alt)):
    plt.vlines(v_alt[j],ymin=0,ymax=entradas_alt[i_alpha_alt-1],color='violet')

# Marco el a correspondiente a la significancia en verde
plt.axvline(alpha,color='yellowgreen',linewidth=3,label='a');

plt.xlabel('t');
plt.ylabel('Densidad de probabilidad f(t|H1)')
plt.legend(loc='upper right');
fig = plt.gcf()
fig.set_size_inches(8,6)
```



In [18]:

```
# Acá solo grafico las dos distribuciones juntas marcando el valor de "a" para que se
vea visualmente la potencia del test.

ax1 = plt.subplot(311)
e, bi, p = plt.hist(t_est,bins=bines,density=True,label='Cuentas',color='rebeccapurpl
e');
plt.errorbar(bin_centers, entradas, yerr, fmt='m.')
plt.setp(ax1.get_xticklabels(), fontsize=10);
plt.axvline(alpha,color='yellowgreen',linewidth=3,label='a');

i_alpha = np.digitize(alpha,bins)

for i in range(0,i_alpha):
    p[i].set_facecolor('rebeccapurple')
for i in range(i_alpha,bines):
    p[i].set_facecolor('violet')
v = np.arange(alpha, bi[i_alpha],0.0001)
plt.vlines(v[0],ymin=0,ymax=e[i_alpha-1],color='violet',label='Significancia')
for j in range(len(v)):
    plt.vlines(v[j],ymin=0,ymax=e[i_alpha-1],color='violet')
plt.xlabel('t');
plt.ylabel('Densidad de probabilidad f(t|H0)')
plt.legend(loc='upper right');

ax2 = plt.subplot(312, sharex=ax1)
e_alt, bi_alt, p_alt = plt.hist(t_est_alt,bins=bines_alt,density=True,label='Cuentas'
,color='rebeccapurple');
plt.errorbar(bin_centers_alt, entradas_alt, yerr_alt, fmt='m.')
plt.setp(ax2.get_xticklabels(),fontsize=10);
plt.axvline(alpha,color='yellowgreen',linewidth=3,label='a');
v_alt = np.arange(alpha, bi_alt[i_alpha_alt],0.0001)
plt.vlines(v_alt[0],ymin=0,ymax=e_alt[i_alpha_alt-1],color='violet',label='Potencia d
el test')
for j in range(len(v_alt)):
    plt.vlines(v_alt[j],ymin=0,ymax=e_alt[i_alpha_alt-1],color='violet')

for i in range(0,i_alpha_alt):
    p_alt[i].set_facecolor('rebeccapurple')
for i in range(i_alpha_alt,bines_alt):
    p_alt[i].set_facecolor('violet')

plt.xlabel('t');
plt.ylabel('Densidad de probabilidad f(t|H1)')
plt.legend(loc='upper right');

fig = plt.gcf()
fig.set_size_inches(12,10)
```



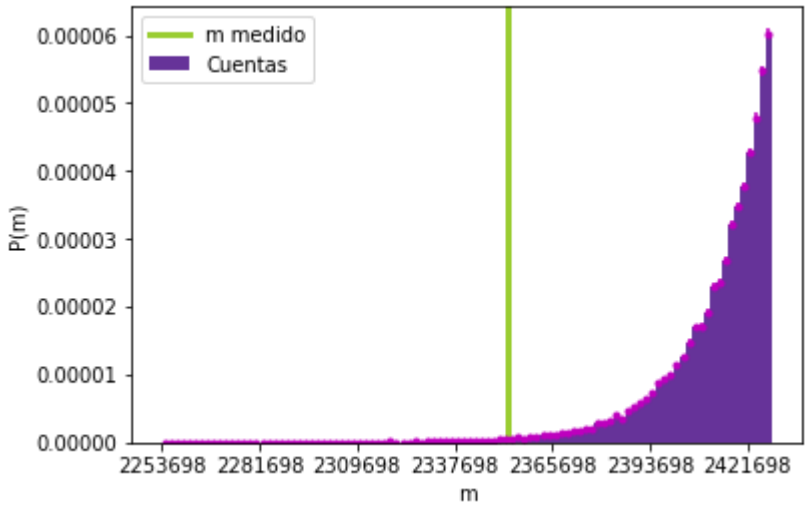
4. La patente del auto más nuevo

In [19]:
```
# Defino (y re defino) valores que voy a necesitar para este ejercicio:
m_medido = max(lista_de_enteros) # Entero correspondiente a la patente más nueva que
 vi
k = len(lista_de_enteros) # Cantidad de patentes que vi

pat_ultima = ['A','D',5,9,2,'M','F']
if pat_ultima in todas_las_patentes:
        n = todas_las_patentes.index(pat_ultima) # Entero correspondiente a la patent
e más nueva en circulación

pat_primera = ['A','D',2,0,0,'A','A']
if pat_primera in todas_las_patentes:
        g = todas_las_patentes.index(pat_primera) # Entero correspondiente a la prime
r patente a considerar
```

4.1 Distribución P(m|n,k) con simulaciones

In [20]:
```
# El procedimiento que sigo para este item es el mismo que en el ejercicio 3: simulo
 conjuntos de datos, calculo el estadístico
# y hago un histograma que me dé su distribución.
# En este caso el estadístico será el valor máximo de los enteros random que tome con
distribución uniforme.

# Genero N valores de m a partir de N conjuntos de datos simulados
m = [] # Vector donde voy a guardar los máximos de cada simulación
N = 50000
for j in range(N):
    W = []
    for i in range (k):
        w = random.randint(0,n)
        W.append(w)
    m.append(max(W))


# Hago el histograma que me va a dar la densidad de probabilidad de m, que es lo que
 me piden.

ent_m, bins_m, patches_m = plt.hist(m,bins=100,density=True,label='Cuentas',color='re
beccapurple');
#Para los errores del histograma
ent__m = np.histogram(m,bins=100)[0]
bin_centers_m = 0.5 * (bins_m[:-1] + bins_m[1:])
bin_width_m = bins_m[1] - bins_m[0]
yerr_m=np.sqrt(ent__m)/(bin_width_m*sum(ent__m))

plt.errorbar(bin_centers_m, ent_m, yerr_m, fmt='m.')
plt.xlabel('t');
plt.axvline(m_medido,color='yellowgreen',linewidth=3,label='m medido'); # Agrego adem
ás el m que yo medí, pero no se pide.
plt.xlabel('m');
plt.ylabel('P(m)');
plt.xticks(np.arange(min(m), max(m), step=28000));
plt.legend(loc='upper left');
print('La distribución del estadístico m está dada por')
```

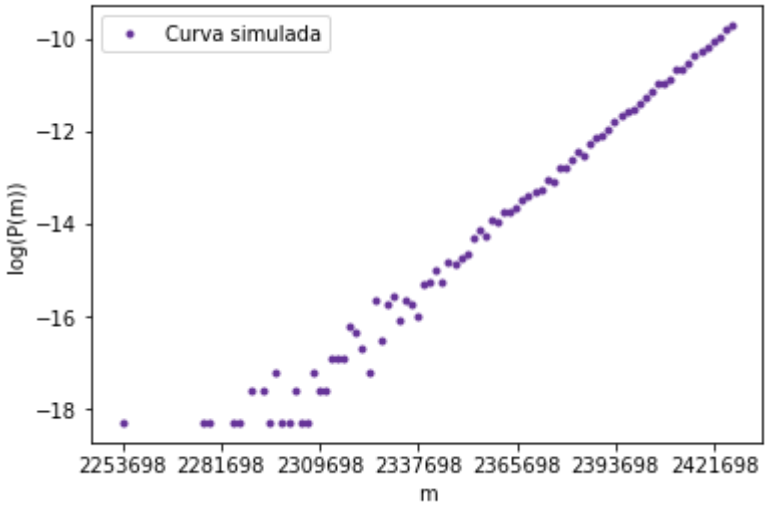
```
La distribución del estadístico m está dada por
```



Esta es la distribución de P(m|k,n) que encuentro a partir de las simulaciones. Para calcularla con la distribución teórica "real", voy a trabajar con el logaritmo de P para evitar los combinatorios de números muy grandes. Por esto, como la distribución de P teórica que voy a tener va a ser log(P), calculo el logaritmo de las entradas del histograma que acabo de calcular.

In [23]:
```
log_ent_m_completo = np.log(ent_m);

log_ent_m_sininf = [] # Me aparecen algunos elementos inf en el vector que me voy a s
acar de encima
bins_m_sininf = []
for j in range(len(log_ent_m_completo)):
    if np.isinf(log_ent_m_completo[j]) == False:
        log_ent_m_sininf.append(log_ent_m_completo[j])
        bins_m_sininf.append(bins_m[j])

# Grafico la curva simulada de log(P(m)) teórica
plt.plot(bins_m_sininf,log_ent_m_sininf,'.',color='rebeccapurple',label='Curva simula
da');
plt.xticks(np.arange(min(m), max(m), step=28000));
plt.xlabel('m');
plt.ylabel('log(P(m))');
plt.legend(loc='upper left');
```

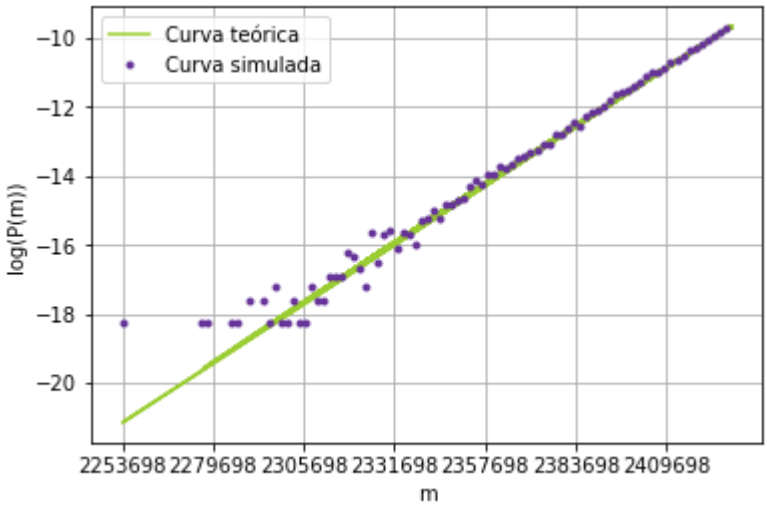
```
C:\Users\Paula\Anaconda2\envs\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: R
untimeWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```



Ahora calculo la curva teórica de log(P) usando la aproximación de Stirling.

In [24]:
```
# Defino la función que voy a aplicar a los valores de m
def Stirling(a):
    s = a*log(a)-a+log(2*np.pi*a)*(1/2)+1/(12*a)
    return s

def Log_P(m,n,k):
    l = Stirling(m-1)-Stirling(m-k)-Stirling(k-1)-Stirling(n)+Stirling(n-k)+Stirling(
k)
    return l
```

In [25]:
```
# Aplico la función
P_m = []
for i in range (len(m)):
    p_m = Log_P(m[i],n,k);
    P_m.append(p_m)

# Grafico ambas curvas juntas

plt.plot(m,P_m,color='yellowgreen',label='Curva teórica');
plt.plot(bins_m_sininf,log_ent_m_sininf,'.',color='rebeccapurple',label='Curva simula
da');
plt.xticks(np.arange(min(m), max(m), step=26000));
plt.xlabel('m');
plt.ylabel('log(P(m))');
plt.legend(loc='upper left');
plt.grid(True)
```



De la comparación de ambas distribuciones puedo decir que la simulación me da una distribución que representa muy bien la real para valores de m grandes, pero no tanto para los valores de m más cercanos a la primera patente considerada.

4.2 Distribución P(n|m,k) con inferencia bayesiana

La forma de calcular la distribución con inferencia bayesiana es utilizando el posterior que se calcula según la siguiente ecuación:

$$P(n|m,k) = \frac{L \cdot \pi(n)}{\int L(m|k,n) \cdot \pi(n) dn}$$

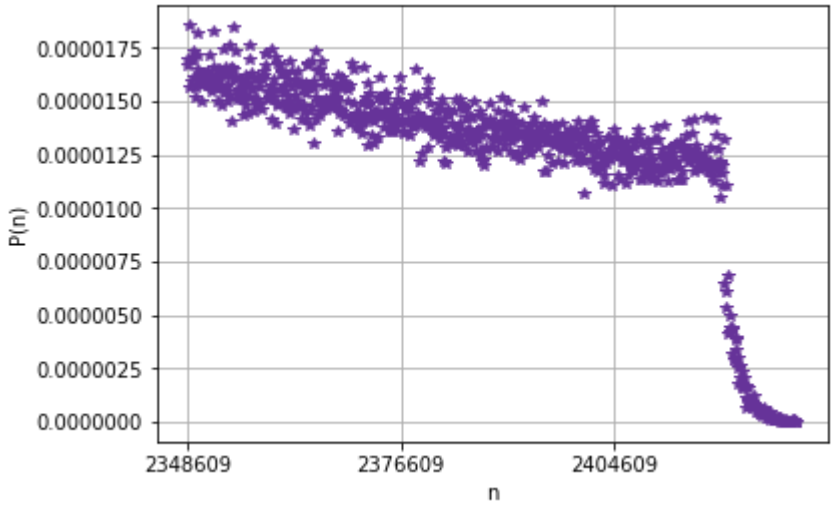
donde $P(n|m,k)$ es lo que llamamos posterior, L es la verosimilitud que se calcula como $L = \prod_{i=0}^{N} P(m_i|k,n)$ y $\pi(n)$ es lo que llamamos prior y debemos nosotros dar según la información previa que tengamos de nuestra distribución.

Entonces: necesito L y π. Me dicen que utilice un prior no informativo, por lo que $\pi = 1$ para cada valor de n. Para calcular L tendría que realizar el producto de todos los valores que toma la densidad $P(m)$ que hallé en el inciso 4.1. Sin embargo, como toma valores muy chicos (del orden de 10^{-9}) y además son muchos datos, esta productoria da un número tan pequeño que es básicamente 0. De cualquier forma, sólo me interesa evaluar en un valor m particular, por lo que voy a usar el $m = n - 10000$ (lo elegí para que sea cercano a n_{real}) en vez de la productoria de todos los m. Haciendo esto obtengo UN valor de $L(n)$, porque está calculado para un n particular. Entonces lo que voy a hacer es repetir el procedimiento N_n veces para tener un vector de valores de L.

In [38]:
```
# Simulo los datos
L_n = [] # Vector donde voy a guardar los valores de L para cada n
todos_los_n = [] # Vector donde voy a guardar los n's que use (me va a servir para el plot).
# En particular, voy a tomar valores de n con pasos de 100 (partiendo del n "real") para acortar la iteración de la simulación.

N_n=800
for r in range(N_n):
    z = 100*r # Defino el paso de n
    n_de_esta_iteracion = n-z # Cambio el n
    m = [] # Misma simulación de datos que antes con menos iteraciones
    N = 1000
    for j in range(N):
        W = []
        for i in range (k):
            w = random.randint(g,n_de_esta_iteracion)
            W.append(w)
        m.append(max(W))

    ent_m, bins_m = np.histogram(m,bins=20,density=True); # Genero el histograma de los máximos correspondiente a esta
    # iteración. Es análogo a la fig 4.1 pero con otro n.

    i_m = np.digitize(n-10000,bins_m)  # Pido el índice del m que elegí
    f_m = []
    if i_m == len(ent_m): # Tuve que poner condicionales para distintos casos de i_m, porque toma todos los valores hasta
            # la longitud de bins_m. Y siendo que los elementos de bins_m son "bordes", tengo más elementos en ese vector que
            # en el vector ent_m que son las entradas de esos bines.
        f_m.append(ent_m[i_m-1])
    elif i_m_medido > len(ent_m):
        f_m.append(ent_m[i_m-2])
    else:
        f_m.append(ent_m[i_m])


    L = np.prod(f_m) # Calculo el valor de la verosimilitud para este n
    L_n.append(L) # Lo guardo en el vector
    todos_los_n.append(n_de_esta_iteracion)
```

In [39]:
```
# Ahora que tengo el vector de L, puedo utilizar el prior = 1 para calcular el posterior (con su respectiva normalización).
# Normalizo dividiendo por la suma de los elementos de L_n y el paso de los n's que tomé.

posterior = L_n/(sum(L_n)*100)

# Grafico la distribución que obtuve

plt.plot(todos_los_n,posterior,'*',color='rebeccapurple');
plt.xticks(np.arange(min(todos_los_n), max(todos_los_n), step=28000))
plt.xlabel('n');
plt.ylabel('P(n)');
plt.grid(True)
print('La distribución del estadístico n está dada por')
```

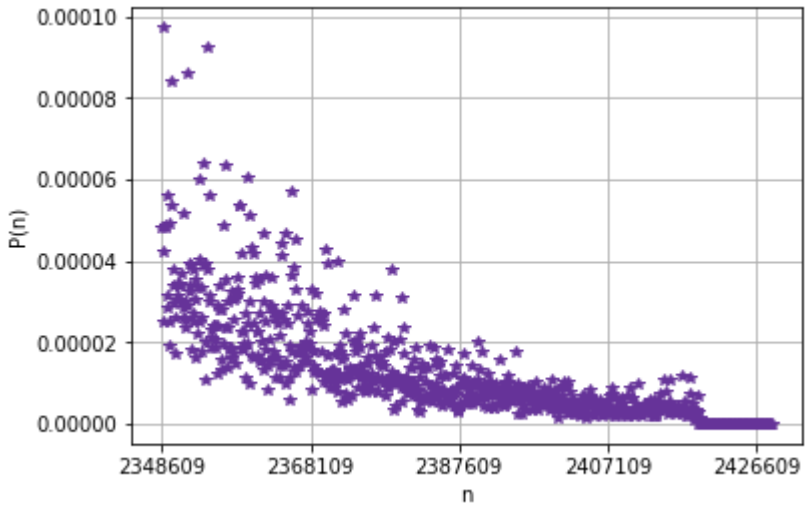
```
La distribución del estadístico n está dada por
```



Esta distribución tiene pinta de ser exponencial. Lo que puedo hacer ahora para tener un posterior (es decir, una densidad de n) aún mejor, es iterar sobre esta última ecuación. Es decir: este posterior va a ser el prior de la siguiente iteración. Y el posterior resultante de ese cálculo será el prior de la siguiente, y así sucesivamente. El posterior se vuelve nuestra información más actualizada de la distribución que queremos, por eso puedo usarla como prior. Entonces itero:

In [42]:
```
posterior = L_n/(sum(L_n)*100)

for h in range(7):
    prior = posterior
    posterior = L_n*prior/sum(L_n*prior*100)

# Grafico la distribución que obtuve

plt.plot(todos_los_n,posterior,'*',color='rebeccapurple');
plt.xticks(np.arange(min(todos_los_n), max(todos_los_n), step=19500))
plt.xlabel('n');
plt.ylabel('P(n)');
plt.grid(True)
print('La distribución del estadístico n después de iterar sobre el posterior está da
da por')
```

```
La distribución del estadístico n después de iterar sobre el posterior está dada por
```



Con las iteraciones, la distribución toma una forma exponencial más evidente.

4.3 Estimación bayesiana para n según mis datos

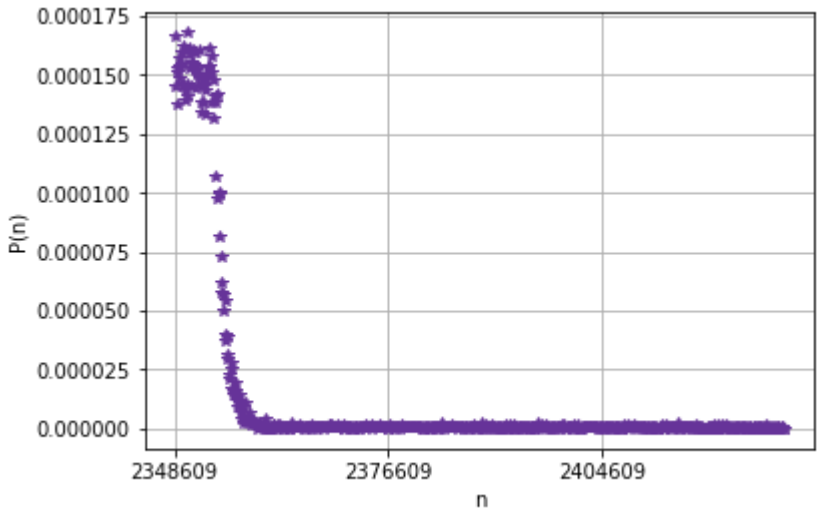
Para calcular mi estimador bayesiano de n tengo que encontrar su esperanza $E = \sum_{i=0}^{N} n \cdot P(n|k,m)$, donde $P(n)$ es el que resulta de usar $m = m_{medido}$ y k mi cantidad total de datos. En la simulación realizada para 4.2 utilicé el k de mis mediciones, pero no el m_{medido}. Entonces repito las simulaciones para este caso y luego calculo la sumatoria correspondiente para hallar el estimador.

In [43]:

```
# Simulo los datos
L_n = [] # Vector donde voy a guardar los valores de L para cada n
todos_los_n = [] # Vector donde voy a guardar los n's que use (me va a servir para el plot).
# En particular, voy a tomar valores de n con pasos de 100 (partiendo del n "real") para acortar la iteración de la simulación.

N_n=800
for r in range(N_n):
    z = 100*r # Defino el paso de n
    n_de_esta_iteracion = n-z # Cambio el n
    m = [] # Misma simulación de datos que antes con menos iteraciones
    N = 1000
    for j in range(N):
        W = []
        for i in range (k):
            w = random.randint(g,n_de_esta_iteracion)
            W.append(w)
        m.append(max(W))

    ent_m, bins_m = np.histogram(m,bins=20,density=True); # Genero el histograma de los máximos correspondiente a esta
    # iteración. Es análogo a la fig 4.1 pero con otro n.

    i_m_medido = np.digitize(m_medido,bins_m)  # Pido el índice del m_medido
    f_m = []
    if i_m_medido == len(ent_m): # Tuve que poner condicionales para distintos casos de i_m, porque toma todos los valores hasta
            # la longitud de bins_m. Y siendo que los elementos de bins_m son "bordes", tengo más elementos en ese vector que
            # en el vector ent_m que son las entradas de esos bines.
        f_m.append(ent_m[i_m_medido-1])
    elif i_m_medido > len(ent_m):
        f_m.append(ent_m[i_m_medido-2])
    else:
        f_m.append(ent_m[i_m_medido])


    L = np.prod(f_m) # Calculo el valor de la verosimilitud para este n
    L_n.append(L) # Lo guardo en el vector
    todos_los_n.append(n_de_esta_iteracion)
```

In [44]:
```
# Ahora que tengo el vector de L, puedo utilizar el prior = 1 para calcular el poster
ior (con su respectiva normalización).
# Normalizo dividiendo por la suma de los elementos de L_n y el paso de los n's que t
omé.

posterior = L_n/(sum(L_n)*100)

# Grafico la distribución que obtuve

plt.plot(todos_los_n,posterior,'*',color='rebeccapurple');
plt.xticks(np.arange(min(todos_los_n), max(todos_los_n), step=28000))
plt.xlabel('n');
plt.ylabel('P(n)');
plt.grid(True)
print('La distribución del estadístico n para m_medido está dada por')
```

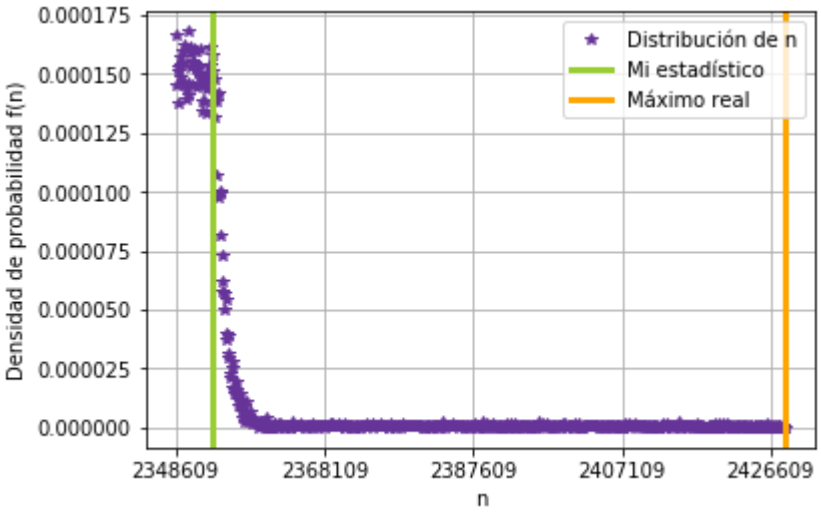
```
La distribución del estadístico n está dada por
```



In [45]:
```
# Ahora para calcular el estimador bayesiano, simplemente hago la sumatoria que corre
sponde de los valores de n y su P(n).

e_n = [] # Guardo cada término de la suma
for j in range (len(todos_los_n)):
    e = todos_los_n[j]*posterior[j]*100
    e_n.append(e)

E_n = sum(e_n) # Sumo todos los elementos

E_n_int = int(E_n) # Me quedo con el entero más cercano

print('Mi estimación bayesiana para n es', todas_las_patentes[E_n_int])
```

```
Mi estimación bayesiana para n es ['A', 'D', 4, 8, 1, 'L', 'P']
```

In [46]:
```
# Grafico la distribución con mi estimador bayesiano
posterior = L_n/(sum(L_n)*100)

plt.plot(todos_los_n,posterior,'*',color='rebeccapurple',label='Distribución de n');
# P(n|k,m)
plt.axvline(E_n_int,color='yellowgreen',linewidth=3,label='Mi estadístico');  # Mi estadístico
plt.axvline(n,color='orange',linewidth=3,label='Máximo real'); # Entero máximo real según el enunciado

plt.xticks(np.arange(min(todos_los_n), max(todos_los_n), step=19500))
plt.xlabel('n');
plt.ylabel('Densidad de probabilidad f(n)');
plt.legend(loc='upper right');
plt.grid(True)
```



4.4 ¿Cuánta suerte tuve el día que armé la lista?

La probabilidad de obtener un estadístico igual o peor que el que obtuve depende de qué tan malo sea mi m_{medido}. Cuanto más lejos está m_{medido} de n, la distribución $P(n|k, m_{medido})$ se aleja más de la correcta, y entonces el estadístico que obtengo es peor. Entonces, como el proceso de medición sigue siendo "observar patentes y ver cuál es mi m", para saber cuánta suerte tuve miro la probabilidad de obtener el m_{medido} o uno peor, es decir, el p-valor de m_{medido}. Integro desde 0 hasta dicho valor de m en la distribución $P(m)$.

In [47]:
```
# Repito el procedimiento de 4.1
m = [] # Vector donde voy a guardar los máximos de cada simulación
N = 50000
for j in range(N):
    W = []
    for i in range (k):
        w = random.randint(0,n)
        W.append(w)
    m.append(max(W))


# Hago el histograma que me va a dar la densidad de probabilidad de m.

ent_m, bins_m, patches_m = plt.hist(m,bins=100,density=True,label='Cuentas',color='rebeccapurple');
plt.clf();
# Calculo el p-valor
i_m_medido = np.digitize(m_medido,bins_m)

ancho_m = m_medido - bins_m[i_m_medido] # ancho de la porcion de bin que tengo que integrar desde que empieza el bin hasta m_medido

# Multiplicando por la altura correspondiente en cada caso y sumando, calculo el p-valor
pvalue_m = bin_width_m * sum(ent_m[0:i_m_medido]) + ent_m[i_m_medido]*ancho_m

print('El p-value para el m medido es', "%.5f" %  pvalue_m)
```

```
El p-value para el m medido es 0.00730
```

```
<Figure size 432x288 with 0 Axes>
```

Dada la pequeña probabilidad que existe de haber medido un m peor que el mio, voy a decir que tuve mucha mala suerte ese día.

No obstante, la probabilidad de obtener un estadístico peor **dado el m que en efecto medí ese día** es calcular el p-valor, pero ahora respecto a mi n estadístico y sobre la distribución $P(n)$:

In [48]:
```
def find_nearest(array, value): # Defino una función que encuentre el valor más cerca
no a uno que yo le doy, de un vector. Esto
    # lo hago porque mi estadístico no pertenece al vector de todos los n que usé.
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]

cercano = find_nearest(todos_los_n,E_n_int)

i_cercano = np.where(todos_los_n == cercano) # Busco el índice de ese valor

areas = [] # Vector donde voy a guardar el área correspondiente a cada rectángulo de
 ancho 1 debajo de la curva P(n)
j=i_cercano[0]
while j<len(posterior): # Este es el barrido de índices que tengo que hacer porque el
vector n va de mayor a menor.
    a = posterior[j]
    areas.append(a*100)
    j = j+1

print('La probabilidad de obtener un estadístico más alejado que el que obtuve del va
lor "real" de la última patente es', sum(areas))
```

```
La probabilidad de obtener un estadístico más alejado que el que obtuve del valor "r
eal" de la última patente es [0.73841402]
```

Obtener un estadístico más alejado del valor "real" de n dado el m_{medido} es muy alta porque dada la forma de la distribución $P(n)$, los valores de n a la izquierda de mi estadístico tienen mayor probabilidad que los que están a la derecha (y son más cercanos al n real). Básicamente, mi m_{medido} es tan inoportuno que la distribución de $P(n)$ que le corresponde me da que los valores cercanos al n_{real} tienen una probabilidad bajísima, básicamente nula. Por esto, de nuevo, tuve una suerte tristísima el día que junté las patentes.

5. ¿Independiente del barrio?

5.1 Test de Wilcoxon

In [49]:
```
# Importo las patentes de Alf
patentes_alf = [['A','A',7,8,9,'G','H'],['A','C',2,0,1,'L','C'],['A','D',2,4,1,'S',
'X'],['A','A',7,3,3,'J','U'],['A','B',6,2,3,'V','C'],['A','C',8,2,3,'X','R'],['A','C'
,0,0,4,'U','U'],['A','B',4,6,0,'U','T'],['A','D',2,0,2,'A','T'],['A','B',9,8,4,'T',
'D'],['A','A',8,1,9,'G','Q'],['A','A',8,4,6,'B','F'],['A','A',9,4,0,'D','Q'],['A','B'
,5,2,2,'V','T'],['A','B',0,5,2,'G','Y'],['A','C',0,4,7,'F','L'],['A','A',3,2,0,'P',
'E'],['A','D',5,6,3,'E','O'],['A','A',5,9,3,'Q','W'],['A','C',6,4,4,'I','J'],['A','B'
,6,4,4,'D','R'],['A','B',9,9,0,'N','J'],['A','A',0,2,0,'H','E'],['A','B',4,8,2,'T',
'U'],['A','C',1,2,1,'K','F'],['A','B',4,4,3,'J','Q'],['A','D',1,2,4,'N','I'],['A','A'
,5,8,5,'N','N'],['A','A',5,4,7,'D','B'],['A','B',1,4,6,'O','I'],['A','A',3,4,0,'J',
'I'],['A','A',5,8,6,'B','B'],['A','D',3,6,1,'P','E'],['A','A',9,6,6,'D','V'],['A','A'
,1,6,6,'M','J'],['A','D',5,3,3,'L','D'],['A','A',1,7,0,'L','H'],['A','C',0,8,0,'R',
'U'],['A','B',8,2,4,'A','K'],['A','A',7,8,5,'K','R'],['A','B',5,7,7,'X','X'],['A','A'
,1,2,6,'I','A'],['A','C',6,1,8,'L','B'],['A','D',0,4,9,'M','Q'],['A','C',9,3,0,'Z',
'E'],['A','B',3,7,9,'G','L'],['A','B',8,8,0,'L','H'],['A','C',3,6,3,'R','P'],['A','C'
,8,4,9,'W','D'],['A','D',0,9,1,'S','T'],['A','A',2,3,9,'K','Y'],['A','D',5,4,1,'B',
'D'],['A','A',0,2,3,'C','Y'],['A','D',4,6,6,'G','K'],['A','B',2,3,4,'Y','T'],['A','A'
,8,3,2,'B','G'],['A','C',0,6,9,'D','L'],['A','B',8,3,9,'C','D'],['A','C',8,6,8,'R',
'M'],['A','A',5,9,9,'H','H'],['A','C',7,0,0,'G','M'],['A','D',4,6,6,'A','W'],['A','C'
,9,7,9,'A','R'],['A','C',2,7,5,'D','G'],['A','A',0,8,2,'K','M'],['A','B',3,1,3,'C',
'A'],['A','B',0,7,4,'A','K'],['A','A',9,5,2,'J','X'],['A','B',6,3,4,'E','Q'],['A','B'
,1,8,6,'I','N'],['A','A',7,3,4,'W','Z'],['A','C',9,4,0,'Y','I'],['A','D',4,7,6,'Z',
'M'],['A','D',3,5,2,'G','H'],['A','C',5,2,0,'R','S'],['A','D',2,1,6,'W','Y'],['A','A'
,6,2,4,'U','P'],['A','D',2,9,9,'F','S'],['A','A',1,3,2,'T','O'],['A','A',1,4,0,'J',
'D'],['A','D',0,8,0,'U','J'],['A','D',4,0,9,'T','A'],['A','C',8,0,0,'V','D'],['A','A'
,1,4,1,'K','J'],['A','A',7,7,5,'C','K'],['A','A',2,3,5,'A','C'],['A','A',1,2,5,'Y',
'S'],['A','C',4,9,1,'U','I'],['A','C',9,2,6,'D','T'],['A','A',1,7,7,'F','K'],['A','C'
,8,9,5,'U','T'],['A','B',7,7,5,'X','V'],['A','A',4,2,7,'X','T'],['A','C',5,2,5,'P',
'H'],['A','D',0,2,9,'I','G'],['A','B',6,6,4,'W','W'],['A','C',2,2,5,'U','S'],['A','C'
,6,2,5,'N','E'],['A','D',0,2,7,'C','D'],['A','C',3,6,9,'C','V'],['A','A',3,3,3,'H',
'O'],['A','D',2,9,9,'G','G'],['A','B',7,4,2,'O','C'],['A','C',2,4,7,'X','D'],['A','B'
,6,8,2,'F','F'],['A','A',4,3,8,'F','L'],['A','B',7,6,6,'H','A'],['A','B',1,5,1,'S',
'N'],['A','C',2,0,6,'A','B'],['A','C',3,3,4,'O','R'],['A','B',5,2,2,'Z','Q'],['A','D'
,1,8,8,'W','R'],['A','A',6,9,4,'T','X'],['A','A',9,8,0,'J','P'],['A','C',7,8,2,'M',
'V'],['A','D',1,4,4,'O','Y'],['A','A',2,4,1,'J','I'],['A','A',8,0,6,'M','X'],['A','B'
,4,0,7,'F','S'],['A','C',1,5,5,'H','K'],['A','A',3,3,1,'D','V'],['A','C',4,1,9,'T',
'P'],['A','A',7,0,6,'G','H'],['A','C',4,2,0,'H','H'],['A','B',0,8,8,'O','I'],['A','B'
,8,1,3,'S','H'],['A','B',5,3,3,'U','L'],['A','B',4,6,1,'E','E'],['A','C',8,0,6,'P',
'V'],['A','B',4,2,8,'J','B'],['A','C',3,1,6,'N','Z'],['A','B',1,5,1,'R','V'],['A','A'
,8,4,1,'B','K'],['A','C',6,1,8,'B','F'],['A','A',9,0,2,'W','M'],['A','B',4,5,1,'Q',
'C'],['A','C',3,2,9,'E','X'],['A','A',1,7,2,'J','Q'],['A','A',7,2,3,'M','P'],['A','B'
,4,6,3,'D','L'],['A','D',2,5,2,'A','T']]
```

In [50]:
```
lista_de_enteros_alf = [] # Vector donde voy a guardar los enteros que le corresponda
n a cada patente que anotó Alf

for j in range(len(patentes_alf)):
    if patentes_alf[j] in todas_las_patentes:
        i_alf = todas_las_patentes.index(patentes_alf[j])
        lista_de_enteros_alf.append(i_alf) # Guardo todos los índices
```

In [51]:
```
# Primero chequeo si tenemos patentes repetidas, porque si no, esto me ahorra problem
as al momento de armar el estadístico de Wilcoxon
for j in range (len(mis_patentes)):
    if mis_patentes[j] in patentes_alf:
        print('Hay una repetida!')

# Como la condición nunca se cumple, nunca printea, entonces no tenemos patentes en c
omún.
```

Entonces, ahora voy a calcular el estadístico de Wilcoxon para testear la hipótesis nula "H_0: Mis mediciones y las de Alf provienen de una distribución con la misma esperanza".

In [52]:
```
# Ahora voy a juntar ambos vectores de enteros en uno solo y le voy a pedir que me los ordene.

lista_de_enteros_total = []
for j in range (len(lista_de_enteros)):
    lista_de_enteros_total.append(lista_de_enteros[j])

for j in range (len(lista_de_enteros_alf)):
    lista_de_enteros_total.append(lista_de_enteros_alf[j])

lde_total = sorted(lista_de_enteros_total)
```

In [53]:
```
# Ahora puedo calcular el estadístico de Wilcoxon. Le voy a pedir que si reconoce algún elemento de mi lista, entonces guarde el
# índice que le corresponde a ese elemento, y luego voy a sumar todos los índices.

indices_mias = []
for j in range(len(lde_total)):
    if lde_total[j] in lista_de_enteros:
        indices_mias.append(j)

W_medido = sum(indices_mias) # Estadístico de Wilcoxon
print('Mi estadístico de Wilcoxon es', W_medido)
```

```
Mi estadístico de Wilcoxon es 22586
```

Ahora que tengo mi estadístico medido, solo necesito la distribución de W para calcular el p-valor como corresponda. Dado que para muestras con más de 10 elementos, ya puedo considerar la distribución del estadístico W como una gaussiana con esperanza

$$E(W) = \frac{n}{2}(n+m+1)$$

y varianza

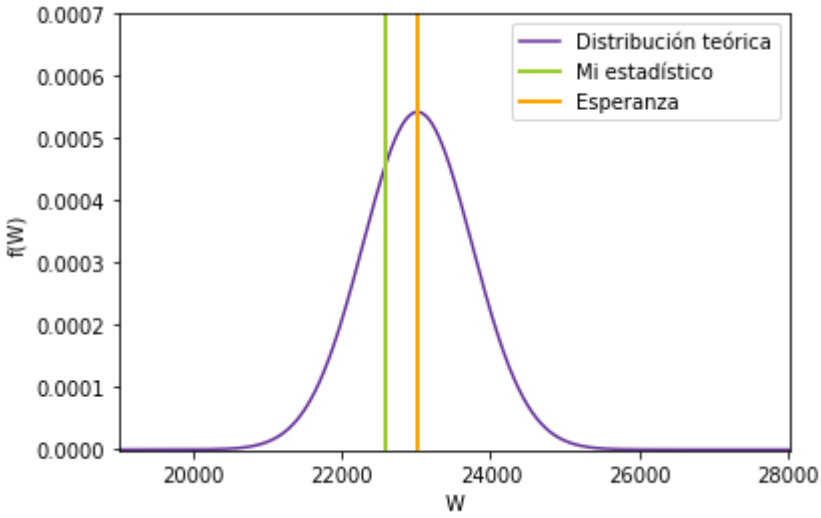
$$V(W) = \frac{nm}{12}(n+m+1)$$

donde n sería la cantidad de elementos en mi muestra y m la cantidad de elementos en la muestra de Alf, uso esta como la distribución de W para calcular el p-valor.

In [54]:
```
# Calculo la esperanza y la varianza para los datos
k = len(lista_de_enteros) # n
k_alf = len(lista_de_enteros_alf) # m

E_w = (k/2)*(k+k_alf+1) # Esperanza
V_w = (k*k_alf/12)*(k+k_alf+1) # Varianza
```

In [55]:
```
# Defino la función gaussiana

def Gaussiana(k,mu,sigma):
    return (1/(np.sqrt(2*np.pi)*sigma))*np.exp((-1/2)*(k-mu)**2/sigma**2)
```

In [56]:
```
# Evalúo la gaussiana en un vector acorde

vector = np.arange(0,max(lde_total)+1)
G = []
for j in range(len(vector)):
    g = Gaussiana(vector[j],E_w,np.sqrt(V_w))
    G.append(g)


# Grafico la distribución junto con su esperanza y el estadístico que calculé

plt.plot(vector,G,color='rebeccapurple',label='Distribución teórica');
plt.axvline(W_medido,color='yellowgreen',linewidth=2,label='Mi estadístico');  # Mi e
stadístico
plt.axvline(E_w,color='orange',linewidth=2,label='Esperanza');  # Esperanza
plt.axis([19000,E_w+5000,-0.000001,0.0007])
plt.xlabel('W');
plt.ylabel('f(W)')
plt.legend(loc='upper right');
```



In [57]:
```
# Ahora tengo todo lo que necesito para calcular el p-valor. En este caso, voy a tene
r que calcular la integral a 2 colas, es
# decir, desde 0 hasta mi estadistico y luego desde el valor que está a igual distanc
ia de la esperanza de la distribución,
# pero del otro lado de la curva, hasta el final.

areas_w = [] # Vector donde voy a guardar el área correspondiente a cada rectángulo d
e ancho 1 debajo de la curva f(W) que
# corresponda integrar

# Integro desde 0 hasta mi estadístico
i_w_medido = np.where(vector == W_medido)

j=0
while j<i_w_medido[0]:
    g = G[j]
    areas_w.append(g)
    j = j+1

# Integro desde el valor que está a igual distancia de la esperanza que mi estadístic
o, pero del otro lado de la curva, hasta el
# final

i_w_medido_bis = np.where(vector == E_w + (E_w-W_medido))

j = int(i_w_medido_bis[0])
while j<len(vector):
    g = G[j]
    areas_w.append(g)
    j = j+1

pvalue_w = sum(areas_w)
print('El p-valor para mi estadístico en el test de Wilcoxon es', pvalue_w)
```

```
El p-valor para mi estadístico en el test de Wilcoxon es 0.5574037214718719
```

Como el W_{medido} está tan cerca de la esperanza, da lugar a un p-valor grande, por lo que puedo decir que la medición que obtuve es muy buena, representando que mis mediciones y las de Alf provienen de la misma distribución (es decir, puedo aceptar la hipótesis nula).

5.2 Distribución del estadístico U

El estadístico propuesto en el ejercicio 4 de la guía 8 para dos sets de datos $X_1, \dots X_n$ e $Y_1, \dots Y_m$ es

$$U = (\bar{X} - \bar{Y}) \sqrt{\frac{m+n-2}{(1/m + 1/n)(s_X^2 + s_Y^2)}}$$

donde $s_X^2 = \sum_i^n (X - \bar{X})^2$ y $s_Y^2 = \sum_i^m (Y - \bar{Y})^2$.

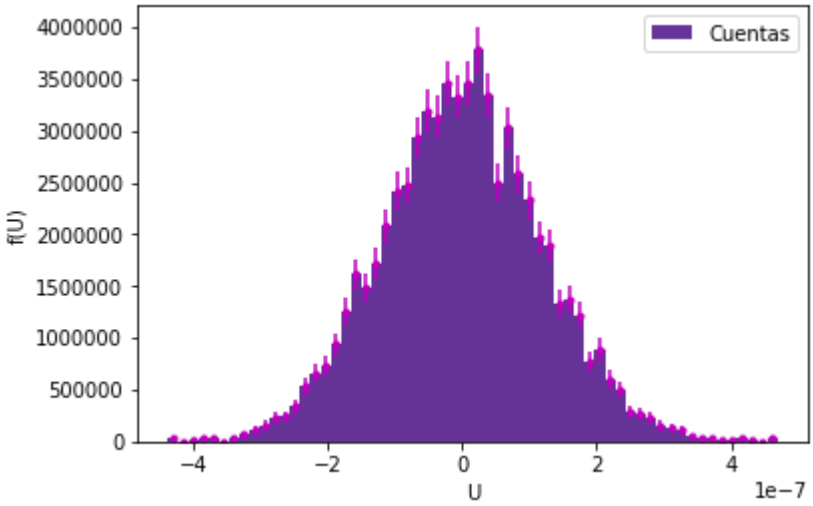
Como en mi caso los datos X e Y provienen de una distribución uniforme (hipótesis que acepté en el ejercicio 3.1) y no gaussiana, no puedo decir que U tenga distribución t-Student. Entonces, tengo que hallar esa distribución. Para eso, voy a hacer lo mismo que en los casos anteriores: simular sets de datos, calcular el estadístico para cada (par de) conjunto de datos y luego hago un histograma.

In [58]:

```
# Simulo 5000 pares de conjuntos de datos: uno de longitud igual que mi vector de dat
os y otro de longitud igual al vector de
# Alf

U = []

for j in range(5000):
    Simul_mias = []
    while len(Simul_mias)<=k:
        r = random.randint(0,n) # distribución uniforme
        if r in Simul_mias:  # Pido esto antes de appendear en mi vector de medicione
s para asegurarme de no repetir enteros
            Simul_mias = Simul_mias
        else:
            Simul_mias.append(r)

    Simul_alf = []
    while len(Simul_alf)<=k_alf:
        t = random.randint(0,n) # distribución uniforme
        if t in Simul_alf:  # Pido esto antes de appendear en mi vector de mediciones
para asegurarme de no repetir enteros
            Simul_alf = Simul_alf
        else:
            Simul_alf.append(t)

    # Calculo el estadístico U

    s_p = []
    for j in range(k):
        s_pp = (Simul_mias[j]-np.mean(Simul_mias))**2
        s_p.append(s_pp)

    s_mias = sum(s_p)


    s_a = []
    for j in range(k_alf):
        s_aa = (Simul_alf[j]-np.mean(Simul_alf))**2
        s_a.append(s_aa)

    s_alf = sum(s_a)

    u = (np.mean(Simul_mias)-np.mean(Simul_alf))*np.sqrt((k+k_alf-2)/((1/k+1/k_alf)*(
s_alf**2 + s_mias**2)))
    U.append(u)
```

In [59]:
```
bines_u = 60
ent_u, bins_u, patches_u = plt.hist(U,bins=bines_u,density=True,label='Cuentas',color
='rebeccapurple');
#Para los errores del histograma
ent__u = np.histogram(U,bins=bines_u)[0]
bin_centers_u = 0.5 * (bins_u[:-1] + bins_u[1:])
bin_width_u = bins_u[1] - bins_u[0]
yerr_u=np.sqrt(ent__u)/(bin_width_u*sum(ent__u))

plt.errorbar(bin_centers_u, ent_u, yerr_u, fmt='m.')
plt.xlabel('t');
plt.xlabel('U');
plt.ylabel('f(U)')
plt.legend(loc='upper right');
print('La distribución del estadístico U está dada por')
```

```
La distribución del estadístico U está dada por
```



La distribución tiene una forma gaussiana, al igual que el estadístico W.

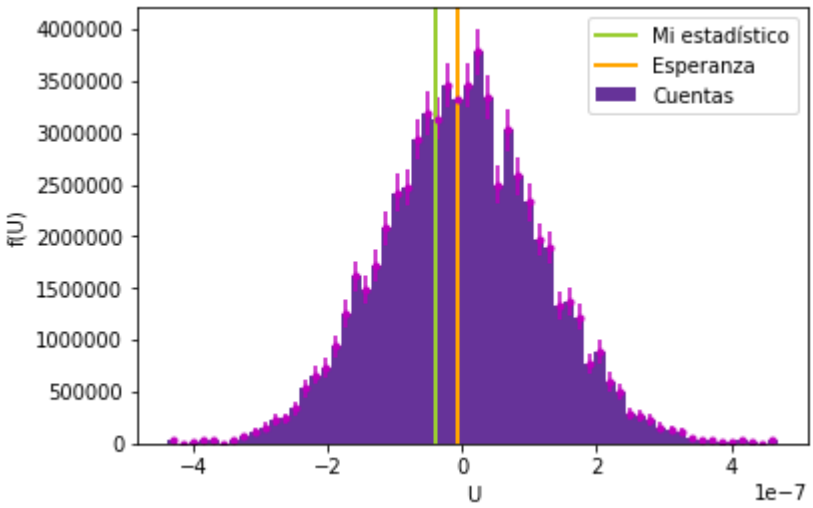
In [60]:
```
# Calculo la esperanza de la distribución porque me va a ser útil después

e_u = []
for j in range(len(ent_u)):
    e = bins_u[j] * ent_u[j] *bin_width_u
    e_u.append(e)

E_u = sum(e_u)
print('La esperanza de la distribución de U es', E_u)
```

```
La esperanza de la distribución de U es -7.745314173938285e-09
```

5.3 Test de estadístico U

In [61]:
```
# Calculo el estadístico U pero ahora usando mis datos reales y los de Alf

s_p = []
for j in range(k):
    s_pp = (lista_de_enteros[j]-np.mean(lista_de_enteros))**2
    s_p.append(s_pp)

s_mias = sum(s_p)


s_a = []
for j in range(k_alf):
    s_aa = (lista_de_enteros_alf[j]-np.mean(lista_de_enteros_alf))**2
    s_a.append(s_aa)

s_alf = sum(s_a)

U_medido = (np.mean(lista_de_enteros)-np.mean(lista_de_enteros_alf))*np.sqrt((k+k_alf
-2)/((1/k+1/k_alf)*(s_alf**2 + s_mias**2)))
```

In [62]:
```
# Grafico la distribución junto con el estadístico medido y su esperanza

ent_u, bins_u, patches_u = plt.hist(U,bins=60,density=True,label='Cuentas',color='reb
eccapurple');
#Para los errores del histograma
ent__u = np.histogram(U,bins=bines_u)[0]
bin_centers_u = 0.5 * (bins_u[:-1] + bins_u[1:])
bin_width_u = bins_u[1] - bins_u[0]
yerr_u=np.sqrt(ent__u)/(bin_width_u*sum(ent__u))

plt.errorbar(bin_centers_u, ent_u, yerr_u, fmt='m.')
plt.axvline(U_medido,color='yellowgreen',linewidth=2,label='Mi estadístico');  # Mi e
stadístico
plt.axvline(E_u,color='orange',linewidth=2,label='Esperanza');  # Esperanza

plt.xlabel('U');
plt.ylabel('f(U)')
plt.legend(loc='upper right');
```



In [63]:
```
# Calculo que el pvalor, de nuevo, integrando a dos colas.

# De 0 a U_medido:
i_u_medido = np.digitize(U_medido,bins_u)

ancho_u = bins_u[i_u_medido]-U_medido

pvalue_u = bin_width_u * sum(ent_u[0:i_u_medido-1]) + ent_u[i_u_medido]*ancho_u


# De E_u hasta el final
i_u_medido_bis = np.digitize(E_u + (E_u-U_medido),bins_u)

ancho_u_bis = bins_u[i_u_medido_bis]-(E_u + (E_u-U_medido))

pvalue_u_ = bin_width_u * sum(ent_u[i_u_medido_bis+1:60]) + ent_u[i_u_medido_bis]*anc
ho_u_bis

print('El p-valor para el u medido es', pvalue_u+pvalue_u_)
```

```
El p-valor para el u medido es 0.746477031133723
```

De nuevo, como obtuve un p-valor grande, puedo aceptar con seguridad la hipótesis nula de que los datos de Alf y los mios provienen de una distribución con una misma esperanza.

7. Sé tu propio verdugo

Voy a plantear la hipótesis nula $\mathbf{H0 = No\ inventé\ los\ datos}$. Si H0 es verdadera, entonces el estadístico m proviene de la distribución $P(m|k,n)$ propuesta. Entonces el ejercicio 4.1 me da la P(m|H0) que necesito. Usando esa distribución, me fijo dónde cae mi m_{medido} y calculo el p-valor, es decir, qué probabilidad había de medir algo más exótico (menos probable). Y la significancia que les voy a pedir que usen va a cumplir ser estrictamente menor que ese p-valor, para que no rechacen la hipótesis nula que planteé (¡porque no los inventé!), es decir, $\alpha < p - valor_m$. Dado que esto ya lo calculé en el ejercicio 4.4, pido:

In [64]:
```
print('Usen significancia alpha <', "%.5f" %  pvalue_m, 'y salven una vida inocente.'
)
```

```
Usen significancia alpha < 0.00730 y salven una vida inocente.
```

Fin del parcial