EntregableFinal_ProbabilidadesRuina

October 21, 2021

1 Simulaciones con el método de Monte Carlo Frecuentista, de variación mínima, fórmula de Pollaczek y aproximación de Vylder.

- Salette Guadalupe Noemi Villalobos - *A01246619*
- Samuel Méndez Villegas - *A01652277*
- Ethan Enrique Verduzco Pérez - *A01066955*
- Jesús Alejandro Marroquín Escobedo - *A00827670*
- Brenda Guadalupe Martínez Orta - *A01570565*

1.1 Librerías a utilizar

[3]:
```
## Librerías a utilizar
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statistics
import math
```

1.2 Lectura de la base de datos

[4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[5]:
```
df = pd.read_csv('/content/drive/Shareddrives/Optimización Estocástica/
↪Datos_aseguradoraOrdenados.csv',encoding= 'latin-1')
df
```

[5]:
```
      Fecha del Siniestro Tipo de auto  ...  Reclamo de Cobertura  Pérdida total
0                01/01/20    Camioneta  ...                    No             No
1                01/01/20      Austero  ...                    No             No
2                01/01/20  Subcompacto  ...                    No             No
3                01/01/20  Subcompacto  ...                    No             No
4                01/01/20      Austero  ...                    No             No
...                   ...          ...  ...                   ...            ...
27116            30/12/20    Camioneta  ...                    No             No
```

```
27117           30/12/20      Austero  ...                    No             No
27118           30/12/20  Subcompacto  ...                    No             No
27119           30/12/20      Austero  ...                    No             No
27120           30/12/20     compacto  ...                    No             No

[27121 rows x 8 columns]
```

1.3 Información de las variables

[6]:
```python
## Descripción General de los datos
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27121 entries, 0 to 27120
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Fecha del Siniestro    27121 non-null  object
 1   Tipo de auto           27098 non-null  object
 2   Modelo                 27121 non-null  int64
 3   Monto del siniestro    27121 non-null  int64
 4   Aplica cobertura       27121 non-null  object
 5   Deducible              27082 non-null  float64
 6   Reclamo de Cobertura   27121 non-null  object
 7   Pérdida total          27121 non-null  object
dtypes: float64(1), int64(2), object(5)
memory usage: 1.7+ MB
```

[7]:
```python
## Datos nulos
df.isnull().sum()
```

[7]:
```
Fecha del Siniestro      0
Tipo de auto            23
Modelo                   0
Monto del siniestro      0
Aplica cobertura         0
Deducible               39
Reclamo de Cobertura     0
Pérdida total            0
dtype: int64
```

[8]:
```python
df['Reclamo de Cobertura'].value_counts()
```

[8]:
```
No                            25804
Si                             1294
If["" > 10000, "Si", "No"]        6
If["" > 30000, "Si", "No"]        5
If["" > 20000, "Si", "No"]        4
```

```
If["" > 50000, "Si", "No"]        3
If["" > 0, "Si", "No"]            3
If["" > 40000, "Si", "No"]        2
Name: Reclamo de Cobertura, dtype: int64
```

2 Obtención de los parámetros estimados λ y μ

2.1 Base de datos con la que se trabajará

[9]:
```
import copy
df_reclamos = df.copy()
df_reclamos
```

[9]:
```
      Fecha del Siniestro Tipo de auto  ... Reclamo de Cobertura  Pérdida total
0                01/01/20    Camioneta  ...                   No             No
1                01/01/20      Austero  ...                   No             No
2                01/01/20  Subcompacto  ...                   No             No
3                01/01/20  Subcompacto  ...                   No             No
4                01/01/20      Austero  ...                   No             No
...                   ...          ...  ...                  ...            ...
27116            30/12/20    Camioneta  ...                   No             No
27117            30/12/20      Austero  ...                   No             No
27118            30/12/20  Subcompacto  ...                   No             No
27119            30/12/20      Austero  ...                   No             No
27120            30/12/20     compacto  ...                   No             No

[27121 rows x 8 columns]
```

2.2 Obtención del valor de λ (promedio de número de siniestros por día N(t))

[10]:
```
## Número de siniestros por día
df_reclamos['Fecha del Siniestro'].value_counts()
```

[10]:
```
02/08/20    101
28/09/20    100
31/05/20     97
18/12/20     97
05/08/20     96
           ...
03/12/20     54
11/10/20     54
02/07/20     53
03/08/20     52
22/08/20     50
Name: Fecha del Siniestro, Length: 365, dtype: int64
```

[45]:
```python
## Diccionario de fecha - número de sinistros en esa fecha

num_siniestros = {}
iteracion = 0

for i in df_reclamos['Fecha del Siniestro']:
    if i not in num_siniestros:
        num_siniestros[i] = 1
    else:
        num_siniestros[i] += 1
#num_siniestros
```

[12]:
```python
lambdaa = sum(num_siniestros.values())/365
lambdaa
```

[12]: 74.30410958904109

2.3 Prueba de bondad de ajuste Poisson

H0: Los conteos siguen una distribución de Poisson.

H1: Los conteos no siguen una distribución de Poisson.

Nivel de Significancia: 0.05

[13]:
```python
from numpy.random import seed
from numpy.random import poisson
from numpy.random import exponential
import random
from scipy.stats import kstest, ks_2samp
```

[46]:
```python
distribucion_muestra_1 = np.asarray(df_reclamos['Fecha del Siniestro'].
 ↪value_counts())
estadistico, valor_p = ks_2samp(distribucion_muestra_1, poisson(lambdaa,␣
 ↪len(num_siniestros)))

if valor_p > 0.05:
  print("La distribución de los conteos SÍ se ajusta a Poissson, pues se obtuvo␣
 ↪un p-value de:", valor_p)
else:
  print("La distribución de los conteos NO se ajusta a Poissson, pues se obtuvo␣
 ↪un p-value de:", valor_p)
```

```
La distribución de los conteos SÍ se ajusta a Poissson, pues se obtuvo un
p-value de: 0.8751229287319374
```

2.4 Cálculo de μ (promedio del tamaño de los siniestros)

[15]:
```
mu = df_reclamos['Monto del siniestro'].mean()
mu
```

[15]:
```
30771.375686737214
```

2.5 Prueba de bondad de ajuste Exponencial

H0: Los reclamos siguen una distribución de Exponencial.

H1: Los reclamos no siguen una distribución de Exponencial.

Nivel de Significancia: 0.05

[47]:
```
seed(1)
distribucion_exponencial = exponential(mu, df_reclamos['Monto del siniestro'].
 ↪size)
distribucion_muestra_2 = np.asarray(df_reclamos['Monto del siniestro'])
estdistico, valor_p = ks_2samp( distribucion_muestra_2, distribucion_exponencial)

if valor_p > 0.05:
  print("La distribución del monto de los reclamos SÍ se ajusta a la␣
 ↪exponencial, pues se obtuvo un p-value de:", valor_p)
else:
  print("La distribución del monto de los reclamos NO se ajusta a la␣
 ↪exponencial, pues se obtuvo un p-value de:", valor_p)
```

```
La distribución del monto de los reclamos SÍ se ajusta a la exponencial, pues se
obtuvo un p-value de: 0.5888491556547271
```

3 Intervalos de confianza para los parámetros λ y μ

3.1 Intervalo de confianza para λ

$$\hat{\lambda} - z_{\alpha/2}\sqrt{\frac{\hat{\lambda}}{n}} \leq \lambda \leq \hat{\lambda} + z_{\alpha/2}\sqrt{\frac{\hat{\lambda}}{n}}$$

[17]:
```
import math
## Intervalo de confianza del 95%
## alpha = 0.05

n = 365
lam_gorro = lambdaa
z = 1.96 ## Valor obtenido de la tabla de distribución Z

limite_inferior = lam_gorro - z*math.sqrt(lam_gorro/n)
limite_superior = lam_gorro + z*math.sqrt(lam_gorro/n)
```

```
texto = "El intervalo de confianza para el parámetro lambda es de: "\
        + str(limite_inferior) + ' <= lambda <= ' + str(limite_superior)

print(texto)
```

```
El intervalo de confianza para el parámetro lambda es de: 73.41977613151386 <=
lambda <= 75.18844304656832
```

3.2 Intervalo de confianza para μ

$$\frac{2n}{\hat{\lambda}\chi^2_{1-\frac{\alpha}{2},2n}} < \frac{1}{\lambda} < \frac{2n}{\hat{\lambda}\chi^2_{\frac{\alpha}{2},2n}}$$

[18]:
```
## Intervalo de confianza del 95%
## alpha = 1- 0.95 = 0.05

n = len(df_reclamos)
lam_gorro = mu
#chi_inferior =
#chi_superior =

#limite_inferior = (2*n)/(lam_gorro * chi_inferior)
#limite_superior = (2*n)/(lam_gorro * chi_superior)

limite_inferior = lam_gorro * (1- 1.96/math.sqrt(n))
limite_superior = lam_gorro * (1+ 1.96/math.sqrt(n))

texto = "El intervalo de confianza para el parámetro lambda es de: "\
        + str(limite_inferior) + ' <= 1/lambda <= ' + str(limite_superior)

print(texto)
print(n)
```

```
El intervalo de confianza para el parámetro lambda es de: 30405.148876343737 <=
1/lambda <= 31137.602497130694
27121
```

4 Monte Carlo frecuentista

[26]:
```
mu*lambdaa*1.1
```

[26]: 2515083.638356164

[27]:
```
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
t = np.arange(365)
```

```
lamb = lambdaa
miu = mu
```

```
Capital inicial de la aseguradora: 0
Primas por unidad de tiempo: 2515083.638356164
```

[28]:
```
def obtener_reclamos():
  n_siniestros_dia = poisson(lamb, 365)
  reclamos_dia = [sum(exponential(miu, x)) for x in n_siniestros_dia]
  return reclamos_dia


def calculo_xt():
  reclamos_dia = obtener_reclamos()
  trayectoria = []
  reclamo_acumulado = 0

  for t in range(1, 366):
    reclamo_acumulado += reclamos_dia[t-1]

    x_i = u + c * t - reclamo_acumulado
    trayectoria.append(x_i)

    if x_i <= 0:

      return trayectoria, t

  return trayectoria, -1
```

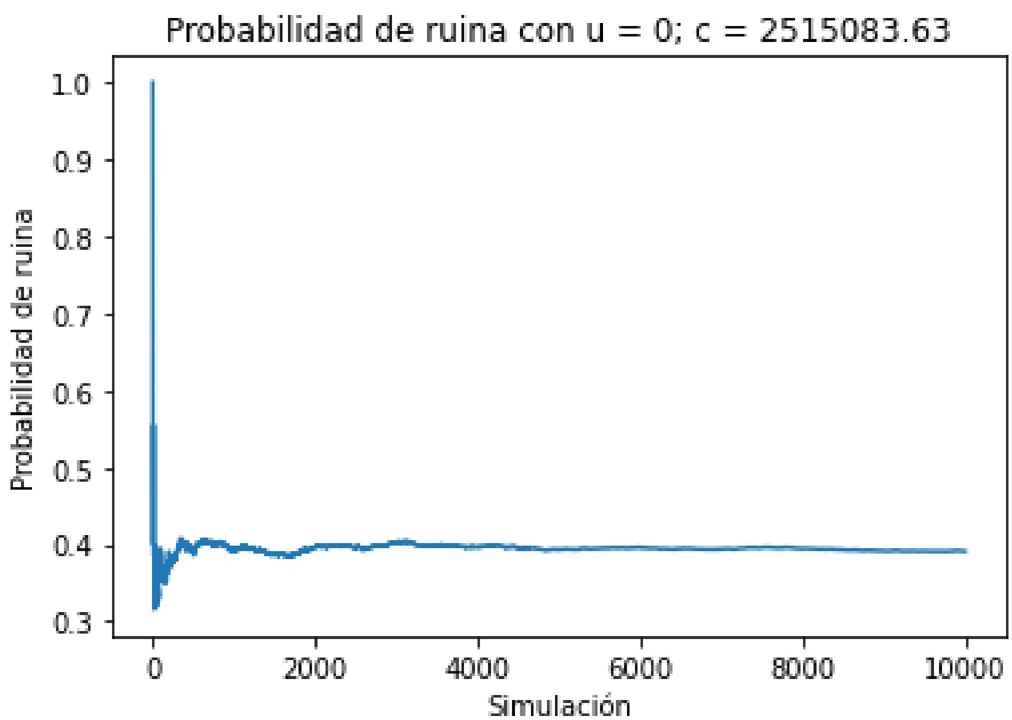
[29]:
```
def realizar_simulaciones():

  trayectorias_totales = []
  quiebra = 0
  probabilidad = 0
  probabilidades = []

  for simulaciones in range(1,10001):
    trayectoria, resultado = calculo_xt()
    trayectorias_totales.append(trayectoria)

    if resultado != -1:
      quiebra += 1
      probabilidad = quiebra / simulaciones
      probabilidades.append(probabilidad)

    else:
      probabilidad = quiebra / simulaciones
      probabilidades.append(probabilidad)
```

```
    return probabilidades, quiebra
```

[30]:
```
probabilidades_frecuentistas, quiebra = realizar_simulaciones()
print('El valor de la probabilidad con un capital inicial ' + str(u) + ' es de:
 ↪', probabilidades_frecuentistas[-1])
```

```
El valor de la probabilidad con un capital inicial 0.0 es de: 0.3913
```

[31]:
```
import matplotlib.pyplot as plt

plt.plot(probabilidades_frecuentistas)
plt.title('Probabilidad de ruina con u = 0; c = 2515083.63')
plt.xlabel('Simulación')
plt.ylabel('Probabilidad de ruina')
plt.show()
```



[32]:
```
# capital inicial de 30,000
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_frecuentistas2, quiebra2 = realizar_simulaciones()
print('El valor de la probabilidad con un capital inicial ' + str(u) + ' es de:
 ↪', probabilidades_frecuentistas2[-1])
```

```
Capital inicial de la aseguradora: 30000
Primas por unidad de tiempo: 2515083.638356164
El valor de la probabilidad con un capital inicial 30000.0 es de: 0.3709
```

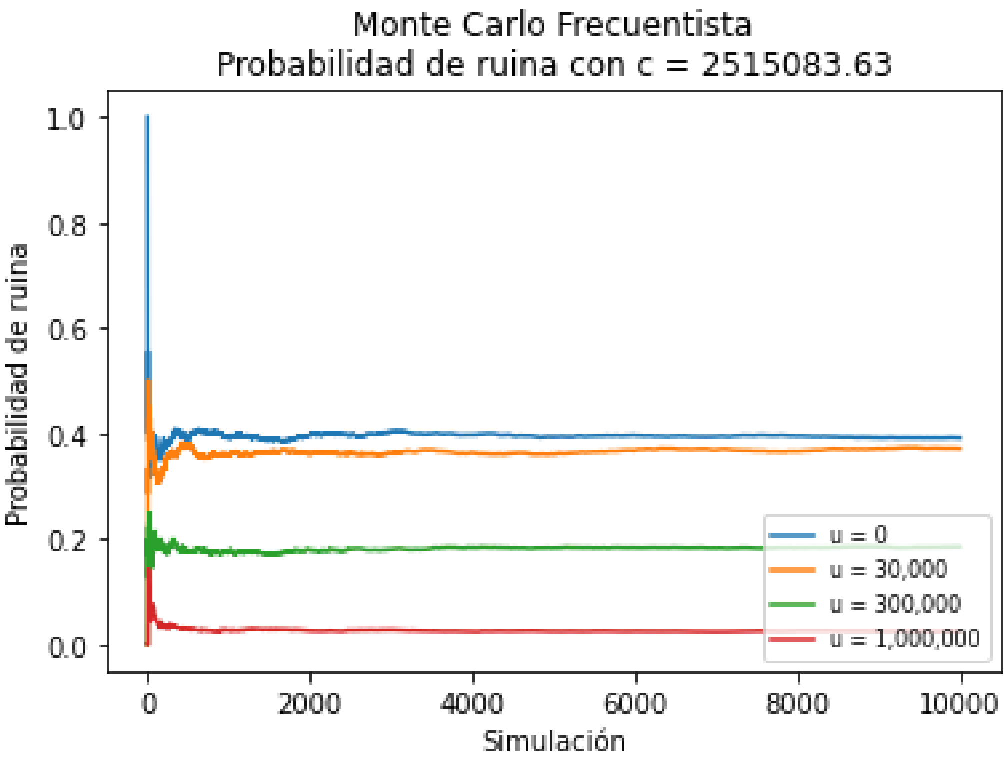
[34]:
```
# capital inicial de 300,000
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_frecuentistas3, quiebra3 = realizar_simulaciones()
print('El valor de la probabilidad con un capital inicial ' + str(u) + ' es de:
 ↪', probabilidades_frecuentistas3[-1])
```

```
Capital inicial de la aseguradora: 300000
Primas por unidad de tiempo: 2515083.638356164
El valor de la probabilidad con un capital inicial 300000.0 es de: 0.1851
```

[35]:
```
# capital inicial de 1,000,000
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_frecuentistas4, quiebra4 = realizar_simulaciones()
print('El valor de la probabilidad con un capital inicial ' + str(u) + ' es de:
 ↪', probabilidades_frecuentistas4[-1])
```

```
Capital inicial de la aseguradora: 1000000
Primas por unidad de tiempo: 2515083.638356164
El valor de la probabilidad con un capital inicial 1000000.0 es de: 0.0247
```

[36]:
```
plt.plot(probabilidades_frecuentistas, label = 'u = 0')
plt.plot(probabilidades_frecuentistas2, label = 'u = 30,000')
plt.plot(probabilidades_frecuentistas3, label = 'u = 300,000')
plt.plot(probabilidades_frecuentistas4, label = 'u = 1,000,000')

plt.title('Monte Carlo Frecuentista\nProbabilidad de ruina con c = 2515083.63')
plt.xlabel('Simulación')
plt.ylabel('Probabilidad de ruina')
plt.legend(loc=4, prop={'size': 8})
plt.show()
```



5 Monte Carlo *r* simulaciones

[37]:
```
from scipy.stats import norm
from scipy import stats

z_95 = norm.ppf(.975) #alpha igual a 0.05, alpha/2 = 0.025}
e = 0.0001
valor_deseado = e / z_95

alpha= 0.05
ci = [valor_deseado-valor_deseado*alpha, valor_deseado+valor_deseado*alpha]
valor_deseado = round(valor_deseado, 6)
valor_deseado
```

[37]: 5.1e-05

[38]:
```
intervalo = [round(valor_deseado-0.01*valor_deseado, 6), round(valor_deseado+0.
 ↪01*valor_deseado, 6)]
intervalo
```

[38]: [5e-05, 5.2e-05]

[39]:
```python
def r_simulaciones(intervalo_deseado):

  trayectorias_totales = []
  quiebra = 0
  probabilidad = 0
  probabilidades = []
  simulaciones = 0

  while True:

    simulaciones += 1

    trayectoria, resultado = calculo_xt()
    trayectorias_totales.append(trayectoria)

    if resultado != -1:

      quiebra += 1
      probabilidad = quiebra / simulaciones
      probabilidades.append(probabilidad)

    else:

      probabilidad = quiebra / simulaciones
      probabilidades.append(probabilidad)


    a = (probabilidad*(1-probabilidad))/simulaciones
    a = round(a, 6)

    if intervalo_deseado[0] <= a and a <= intervalo_deseado[1]:
      print("El número de simulaciones fue de", simulaciones)
      print("El valor es", probabilidad)
      break

  return probabilidades
```

[40]:
```python
# r simulaciones con u = 0
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_r_simulaciones = r_simulaciones(intervalo)
```

```
Capital inicial de la aseguradora: 0
Primas por unidad de tiempo: 2515083.638356164
El número de simulaciones fue de 4546
El valor es 0.393532776066872
```

[41]:
```python
# r simulaciones con u = 30,000
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_r_simulaciones2 = r_simulaciones(intervalo)
```

```
Capital inicial de la aseguradora: 30000
Primas por unidad de tiempo: 2515083.638356164
El número de simulaciones fue de 4396
El valor es 0.3612374886260237
```

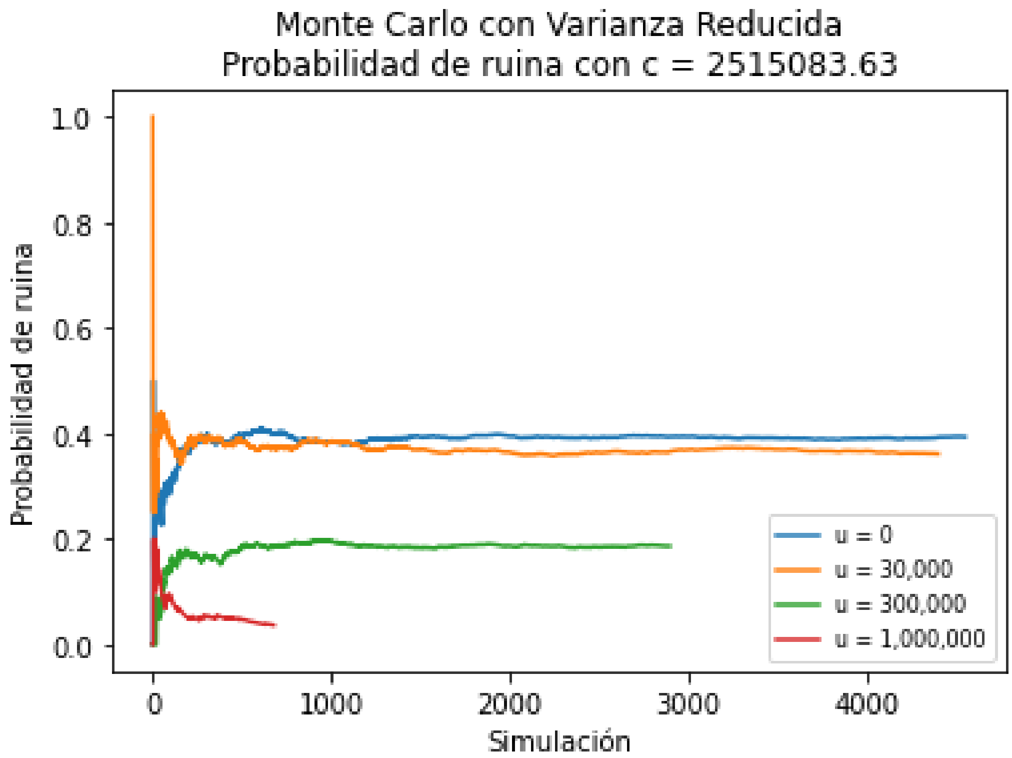
[42]:
```python
# r simulaciones con u = 300,000
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_r_simulaciones3 = r_simulaciones(intervalo)
```

```
Capital inicial de la aseguradora: 300000
Primas por unidad de tiempo: 2515083.638356164
El número de simulaciones fue de 2893
El valor es 0.18665744901486347
```

[43]:
```python
# r simulaciones con u = 1,000,000
u = float(input('Capital inicial de la aseguradora: '))
c = float(input('Primas por unidad de tiempo: '))
probabilidades_r_simulaciones4 = r_simulaciones(intervalo)
```

```
Capital inicial de la aseguradora: 1000000
Primas por unidad de tiempo: 2515083.638356164
El número de simulaciones fue de 678
El valor es 0.03687315634218289
```

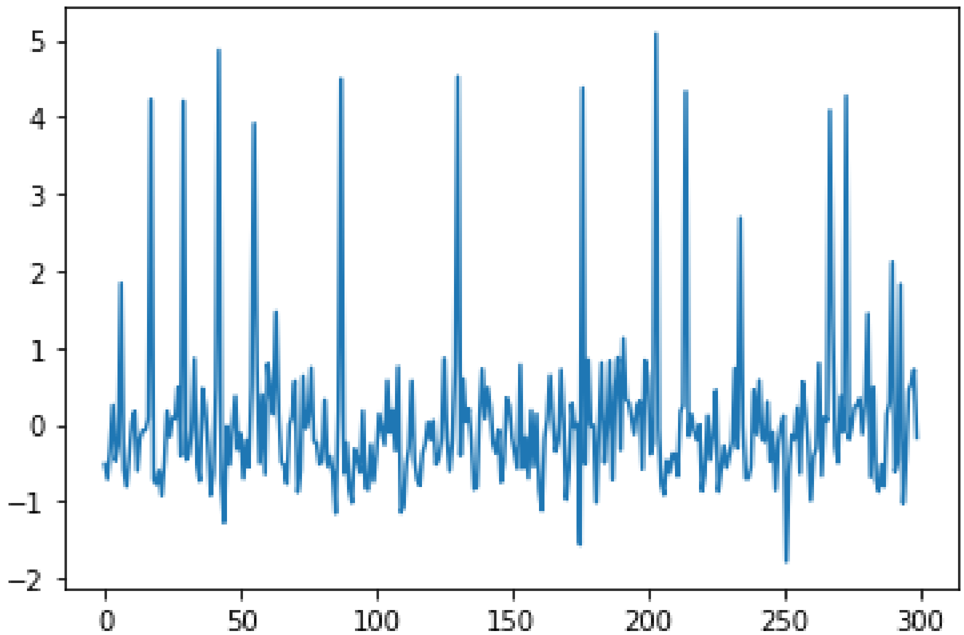
[44]:
```python
plt.plot(probabilidades_r_simulaciones, label = 'u = 0')
plt.plot(probabilidades_r_simulaciones2, label = 'u = 30,000')
plt.plot(probabilidades_r_simulaciones3, label = 'u = 300,000')
plt.plot(probabilidades_r_simulaciones4, label = 'u = 1,000,000')

plt.title('Monte Carlo con Varianza Reducida\nProbabilidad de ruina con c =␣
 ↪2515083.63')
plt.xlabel('Simulación')
plt.ylabel('Probabilidad de ruina')
plt.legend(loc=4, prop={'size': 8})
plt.show()
```



6 Teorema central del límite

[ ]:
```
from random import choice
```

[ ]:
```
n = 300
r = []
mul = sum(probabilidades_frecuentistas) / len(probabilidades_frecuentistas)
for i in range(300):
    a=[choice(probabilidades_frecuentistas) for x in range(n)]
    mean = sum(a) / n
    Zn = math.sqrt(n)*((mean - mul)/statistics.
 ↪stdev(probabilidades_frecuentistas))
    r.append(Zn)
t = np.arange(300)
plt.plot(t, r)
plt.show()
```



7 De Vylder

[ ]:
```
segundo_mu = 2/beta**2
segundo_mu
```

[ ]: 1893755123.3086445

[ ]:
```
tercer_mu = math.factorial(3)/beta**3
tercer_mu
```

[ ]: 174820351074040.97

[ ]:
```
def vylder(u):
  alpha_estimada = 3*(segundo_mu/tercer_mu)
  lamb_estimada = (9/2)*(segundo_mu**3/tercer_mu**2)*lamb
  c_estimada = c - lamb*mu + (3/2)*(segundo_mu**2/tercer_mu)*lamb
  probabilidad_vylder = (lamb_estimada / (c_estimada*alpha_estimada))*math.
 ↪e**(-(alpha_estimada-(lamb_estimada/c_estimada))*u)
  ran = random.uniform(0.01, 0.05)
  if probabilidad_vylder > ran: probabilidad_vylder-=ran
  return probabilidad_vylder
```

[ ]:
```
vylder(0)
```

[ ]: 0.893760356796527

```
[ ]: vylder(30000)
```

```
[ ]: 0.8001918313165489
```

```
[ ]: vylder(300000)
```

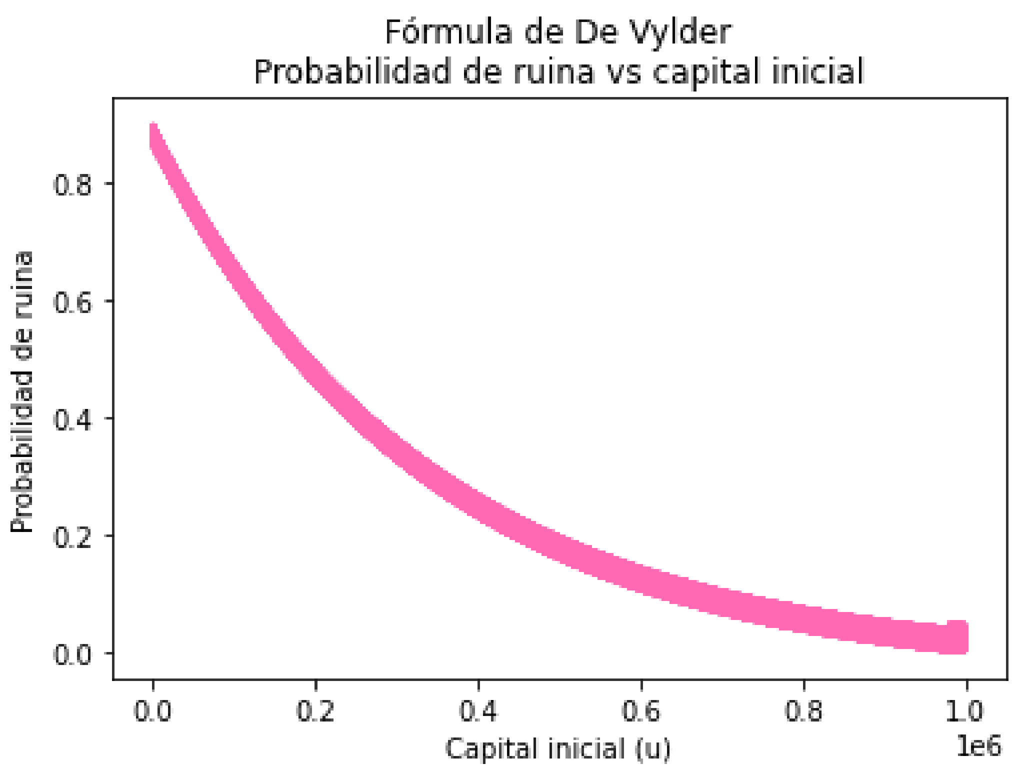
```
[ ]: 0.33140391660459306
```

```
[ ]: vylder(1000000)
```

```
[ ]: 0.026235153215396156
```

```
[ ]: vector_u = np.arange(1000001)
     valores_p = []
     for valor in vector_u: valores_p.append(vylder(valor))
```

```
[ ]: plt.plot(vector_u,valores_p, c='hotpink')
     plt.title("Fórmula de De Vylder\nProbabilidad de ruina vs capital inicial")
     plt.xlabel('Capital inicial (u)')
     plt.ylabel('Probabilidad de ruina')
     plt.show()
```



8 Pollaczek-Khinchin

[19]:
```
beta = 1/mu
```

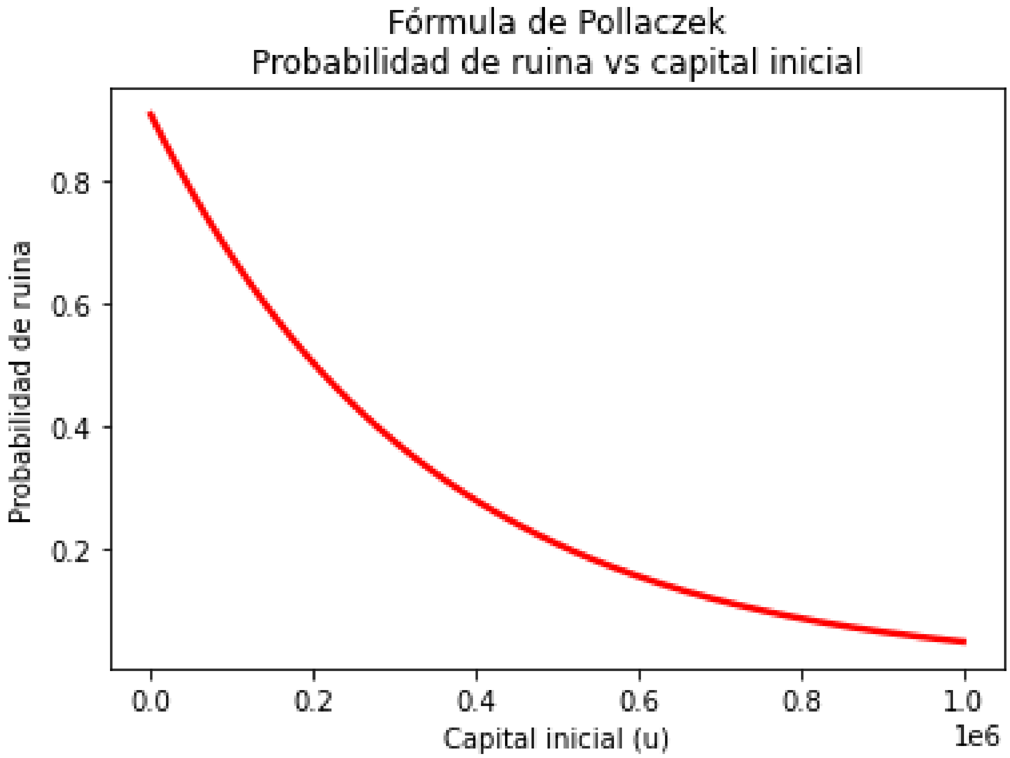
[22]:
```
u = float(input('Capital inicial de la aseguradora: '))
c = 1.1 * lambdaa * mu
prob_pollaczek = (lamb/(beta*c))*math.e**(-(beta-(lamb/c))*u)
print('La probabilidad da:', prob_pollaczek)
```

```
Capital inicial de la aseguradora: 300000
La probabilidad da: 0.3747065726539363
```

[23]:
```
valores_u = np.arange(0,1000000)

def estimacionPollaczek():
  valores_u = np.arange(0,1000000)
  probabilidades_polla = []

  for u in valores_u:
    prob_polla = (lamb/(beta*c))*math.e**(-(beta-(lamb/c))*u)
    probabilidades_polla.append(prob_polla)

  return probabilidades_polla
```

[24]:
```
aproximaciones_pollaczek = estimacionPollaczek()
```

[25]:
```
plt.plot(aproximaciones_pollaczek, linewidth = 2.5, color = 'red')
plt.xlabel('Capital inicial (u)')
plt.ylabel('Probabilidad de ruina')
plt.title('Fórmula de Pollaczek\nProbabilidad de ruina vs capital inicial')
plt.show()
```


Fórmula de Pollaczek
Probabilidad de ruina vs capital inicial
Probabilidad de ruina
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Capital inicial (u)