**1ra Lista Exercícios**

**Nome:** Adrian Alejandro Chavez Alanes  
**Matrícula:** 948

**Link Github:**

1. Considere um Gerador Linear Congruente (GLC) misto com os seguintes parâmetros:
2. Calcule os cinco primeiros números gerador pelo GLC misto.
3. Determine o período desse gerador.

Por definição:

O período máximo de um GLC é no máximo igual ao módulo m.

Por cálculo:

Por código:

|  |
| --- |
| import numpy as np  x = 7  # Valor inicial x0  a = 5  c = 1  m = 16  x1 = [x]  # Cria uma lista con x0 como valor inicial  while True:      x = (a \* x + c) % m  # Gera o próximo número      if x == x1[0]:  # Se repetir o primeiro valor, terminamos          break      x1.append(x)  # Adiciona o novo valor à lista  # Imprimir os valores gerados e o período  print("Valores gerados:", x1)  print("Período:", len(x1)) |

1. Explique se este GLC misto é adequado para aplicações criptográficas. Justifique sua resposta.

R.- Não é adequado porque tem um período muito curto em que um padrão já pode ser encontrado, tornando-o inadequado para aplicações de segurança. Além disso, não é totalmente aleatório, pois depende das variáveis ​​, , e do valor inicial .

1. Em uma central telefônica, o número médio de chamadas recebidas por minuto é igual a 3. Suponha que o número de chamadas recebidas por minuto siga uma distribuição Poisson.
2. Qual é a probabilidade de que exatamente 5 chamadas sejam recebidas em um minuto específico?

Por cálculo:

1. Qual é a probabilidade de que no máximo 2 chamadas sejam recebidas em um minuto específico?

Por cálculo:

Por código (a) y (b):

|  |
| --- |
| import numpy as np  # Definir parámetros  lambda1 = 3  # Número médio de requisições (média da Poisson)  N = 1000000  # Número de amostras (tamanho da simulação)  # Gerar amostras da distribuição de Poisson usando a transformada inversa  av = np.array([])  x = np.random.uniform(0, 1, N)  for ix in x:      i = 0      pr = np.exp(-lambda1)      F = pr      while ix >= F:          pr = lambda1 / (i + 1) \* pr          F = F + pr          i = i + 1      av = np.append(av, i)  # Calcular probabilidades  prob\_2 = np.sum(av == 5) / N  # P(X = 2)  prob\_leq\_2 = np.sum(av <= 2) / N  # P(X ≤ 2)  # Mostrar resultados  print(f"P(X = 2) ≈ {prob\_2:.4f}")  print(f"P(X ≤ 2) ≈ {prob\_leq\_2:.4f}") |

1. Uma prova objetiva possui 10 questões, e cada questão apresenta 4 alternativas, das quais apenas uma é correta. Um aluno despreparado responde aleatoriamente todas as questões, assinalando uma alternativa por questão.

Considere que X seja a variável aleatória que representa o número de questões acertadas pelo aluno.

1. Qual é a probabilidade de o aluno acertar exatamente 3 questões.
2. Qual é a probabilidade de ele acertar no máximo 2 questões.
3. Determine a média e o desvio padrão da variável aleatória X.

Por código:

|  |
| --- |
| import numpy as np  import math  # Parâmetros do problema  n = 10  # Número de perguntas  q = 0.25  # Probabilidade de acerto  # Cálculo da probabilidade de acertar exatamente 3 perguntas  # Combinação para escolher 3 acertos de 10 (nCr)  comb\_3 = math.factorial(n) / (math.factorial(3) \* math.factorial(n - 3))  # Combinação manual  p\_3 = comb\_3 \* (q \*\* 3) \* ((1 - q) \*\* (n - 3))  # Fórmula da binomial para P(X=3)  print(f"P(X=3) = {p\_3:.5f}")  # Exibe a probabilidade de acertar exatamente 3  # Cálculo da probabilidade de acertar no máximo 2 perguntas (P(X ≤ 2))  p\_max\_2 = 0  for k in range(3):  # k vai de 0 até 2 (inclusive)      # Calculando a combinação e a probabilidade de acerto para cada valor de k      comb\_k = math.factorial(n) / (math.factorial(k) \* math.factorial(n - k))      p\_k = comb\_k \* (q \*\* k) \* ((1 - q) \*\* (n - k))  # Probabilidade para um valor k específico      p\_max\_2 += p\_k  # Somando as probabilidades para P(X≤2)  print(f"P(X≤2) = {p\_max\_2:.5f}")  # Exibe a probabilidade de acertar no máximo 2  # Cálculo da média (μ) e da desvio padrão (σ) para a distribuição binomial  media = n \* q  # Média de uma distribuição binomial  varianza = n \* q \* (1 - q)  # Variância da distribuição binomial  desviacao\_estandar = np.sqrt(varianza)  # Desvio padrão  # Exibindo os resultados  print(f"Media (μ) = {media:.2f}")  print(f"Desvio padrão (σ) = {desviacao\_estandar:.5f}") |

1. Se ocorrerem falhas de energia elétrica de acordo com uma distribuição de Poisson com uma média de 6 falhas a cada duas semanas, calcule a probabilidade de que haverá ao menos 3 falhas durante uma semana específica. Traçar o histograma da variável analisada.

Por cálculo:

Por código:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # Definir parámetros  lambda1 = 3  # Média de falhas por semana  N = 1000  # Número de amostras  # Lista para armazenar os valores gerados  av = np.array([])  # Gerar N valores aleatórios uniformemente distribuídos entre 0 e 1  x = np.random.uniform(0, 1, N)  # Aplicar o método da Transformada Inversa para gerar valores Poisson  for ix in x:      i = 0  # Contador de falhas      pr = np.exp(-lambda1)  # Primeiro termo da distribuição de Poisson      F = pr  # Inicializa a soma acumulativa da CDF        # Encontrar o menor i tal que F >= U (inversão da CDF)      while ix >= F:          pr = lambda1 / (i + 1) \* pr  # Atualiza a probabilidade do próximo valor de i          F = F + pr  # Atualiza a soma acumulada da CDF          i = i + 1  # Incrementa o número de falhas        av = np.append(av, i)  # Armazena o número de falhas gerado  # Calcular P(X >= 3) = 1 - P(X <= 2)  prob\_X\_geq\_3 = np.sum(av >= 3) / N  # Imprimir resultado da probabilidade  print(f"Probabilidade de pelo menos 3 falhas em uma semana: {prob\_X\_geq\_3:.4f}")  # Graficar histograma  plt.hist(av, bins=np.arange(0, max(av.astype(int)) + 2) - 0.5, density=True, alpha=0.7, color='b', edgecolor='black')  plt.xlabel("Número de falhas por semana")  plt.ylabel("Frequência relativa")  plt.title("Distribuição de Poisson (λ = 3)")  plt.xticks(range(0, int(max(av)) + 1))  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.show() |

1. O tempo (em minutos) entre chegadas sucessivas de clientes a um caixa eletrônico pode ser descrito por uma variável aleatória com distribuição exponencial, cuja média é de 2 minutos.
2. Qual é o parâmetro dessa distribuição exponencial?
3. Qual é a probabilidade de que o tempo de espera até a chegada do próximo cliente seja inferior a 1 minuto?
4. Qual é a probabilidade de que o tempo de espera até a chegada do próximo cliente seja superior a 4 minutos?

Por código:

|  |
| --- |
| import numpy as np  # Parâmetro da distribuição exponencial  media = 2  # Média da distribuição  lmbda = 1 / media  # Parâmetro lambda  # Cálculo das probabilidades  x1 = 1  p\_x1 = 1 - np.exp(-lmbda \* x1)  print(f"Probabilidade de esperar menos de {x1} minuto: P(X ≤ {x1}) = {p\_x1:.4f}")  x3 = 3  p\_x3 = 1 - np.exp(-lmbda \* x3)  # P(X ≤ 3)  p\_x2 = 1 - p\_x3  # P(X ≥ 4)  print(f"Probabilidade de esperar mais de 4 minutos: P(X ≥ 4) = {p\_x2:.4f}") |

1. A distribuição discreta geométrica conta o número de tentativas até o primeiro sucesso. A pmf é dada por , onde representa a probabilidade de sucesso e o número de tentativas. Fazer um algoritmo para a geração das variáveis aleatórias geométricas. Com o algoritmo proposto calcular:

Um jogador participa de um jogo no qual ele lança um dado justo (equilibrado, com 6 faces numeradas de 1 a 6). Ele ganha o jogo assim que o número "5" aparecer pela primeira vez. Considere que os lançamentos são independentes.

Seja X a variável aleatória que representa o número do lançamento no qual o jogador obtém pela primeira vez o número "5".

Dados:

1. Qual é a probabilidade de que o jogador ganhe o jogo exatamente no terceiro lançamento?
2. Qual é a probabilidade de que ele precise lançar o dado pelo menos 4 vezes para ganhar o jogo?
3. Calcule a média e o desvio padrão de X.

Média:

Desvio padrão:

|  |
| --- |
| import random  # Para gerar números aleatórios  import math    # Para cálculos matemáticos (raiz quadrada)  # Configurações do problema do dado  p = 1/6        # Probabilidade de sucesso (sair o número 5)  num\_simulacoes = 100000  # Número de repetições para as simulações  # Gerando uma variável geométrica (sem usar funções)  intentos = 1  # Começa no primeiro lançamento  # Enquanto não obtivermos sucesso (número aleatório > p)  while random.random() > p:      intentos += 1  # Conta mais um lançamento  valor\_geometrico = intentos  # Guarda o resultado  # Inciso a) Probabilidade P(X=3) - Ganhar no terceiro lançamento exatamente  contador\_a = 0  # Contador para casos onde X=3  for \_ in range(num\_simulacoes):      intentos = 1  # Reinicia contagem      # Simula lançamentos até obter sucesso      while random.random() > p:          intentos += 1      # Se conseguiu no 3º lançamento, conta      if intentos == 3:          contador\_a += 1  # Calcula probabilidade como (casos favoráveis)/(total de tentativas)  prob\_a\_sim = contador\_a / num\_simulacoes  print(f"a) Probabilidade de ganhar no 3° lançamento: {prob\_a\_sim:.4f}")  # Inciso b) Probabilidade P(X≥4) - Necessitar pelo menos 4 lançamentos para ganhar  contador\_b = 0  # Contador para casos onde X≥4  for \_ in range(num\_simulacoes):      intentos = 1      while random.random() > p:          intentos += 1      if intentos >= 4:  # Conta se precisou de 4+ lançamentos          contador\_b += 1  prob\_b\_sim = contador\_b / num\_simulacoes  print(f"b) Probabilidade de precisar de ≥4 lançamentos: {prob\_b\_sim:.4f}")  # Inciso c) Cálculo da média e desvio padrão #  valores = []  # Armazenará todos os resultados das simulações  for \_ in range(num\_simulacoes):      intentos = 1      while random.random() > p:          intentos += 1      valores.append(intentos)  # Guarda cada resultado  # Cálculo da média (soma todos valores e divide pelo total)  media\_sim = sum(valores)/num\_simulacoes  # Cálculo da variância (média dos quadrados das diferenças)  varianza\_sim = sum((x - media\_sim)\*\*2 for x in valores)/num\_simulacoes  # Desvio padrão é a raiz quadrada da variância  desviacion\_sim = math.sqrt(varianza\_sim)  print(f"c) Média estimada: {media\_sim:.2f} lançamentos")  print(f"   Desvio padrão estimado: {desviacion\_sim:.2f} lançamentos") |

1. Utilizando o método da inversa gerar amostrar para a distribuição:

Plotar a pdf analítica e o histograma normalizado.

Cálculos:

Por código:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # Número de amostras  N = 10000  # Passo 1: Gerar valores uniformes U entre 0 e 1  U = np.random.uniform(0, 1, N)  # Passo 2: Aplicar a transformada inversa  X = U\*\*(1/3)  # F^-1(U) = U^(1/3)  # Criar pontos para a PDF analítica  x\_vals = np.linspace(0, 1, 1000)  pdf\_vals = 3 \* x\_vals\*\*2  # f(x) = 3x^2  # Plotar o histograma normalizado  plt.hist(X, bins=40, density=True, alpha=0.6, color='b', edgecolor='black', label='Amostras geradas')  # Plotar a PDF analítica  plt.plot(x\_vals, pdf\_vals, 'r-', lw=2, label='PDF Analítica')  # Configurações do gráfico  plt.xlabel("x")  plt.ylabel("Densidade de probabilidade")  plt.title("Amostras geradas vs. PDF Analítica")  plt.legend()  plt.grid()  plt.show() |

1. Considere uma variável aleatória contínua X cuja função densidade de probabilidade (pdf) é dada por:

Suponha que você queira gerar valores dessa variável usando o **método da aceitação-rejeição.**

1. Verifique que é uma densidade válida.

Uma função deve satisfazer:

1. Não negatividade:
2. Normalização: A integral de no seu domínio deve ser
3. Encontre uma constante c adequada para a aplicação do método da aceitação-rejeição, considerando a distribuição candidata escolhida.

Se deve satisfazer:

Uma escolha comum para é a distribuição uniforme :

O valor máximo é :

1. Explique o procedimento passo a passo para gerar uma observação de X usando esse método.
2. Escolha da distribuição :

é escolhido e a variável é calculada (isso foi feito anteriormente).

1. Geração de um possível :

Geramos um número aleatório e definimos , pois segue a distribuição uniforme

1. Geração de um critério de aceitação:

Geramos outro número aleatório , que será usado para decidir se aceitamos ou não .

1. Critério de aceitação

A razão:

Aceitamos como mostra válida se , caso contrário descartamos e voltamos ao passo 2.

Plotar a pdf analítica e o histograma normalizado

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # Número de amostras  n\_amostras = 10000  amostras = []  # Gerar amostras usando o método da aceitação-rejeição  while len(amostras) < n\_amostras:      U1 = np.random.uniform(0, 1)  # Gerar U1      U2 = np.random.uniform(0, 1)  # Gerar U2        if U2 <= U1\*\*2:  # Critério de aceitação          amostras.append(U1)  amostras = np.array(amostras)  # Gerar valores para a PDF teórica  x\_vals = np.linspace(0, 1, 1000)  pdf\_vals = 3 \* x\_vals\*\*2  # Plotar histograma e PDF teórica  plt.figure(figsize=(8, 5))  plt.hist(amostras, bins=30, density=True, alpha=0.6, color='b', label="Histograma normalizado")  plt.plot(x\_vals, pdf\_vals, 'r-', lw=2, label="PDF analítica $f(x) = 3x^2$")  plt.xlabel("x")  plt.ylabel("Densidade de probabilidade")  plt.title("Histograma de amostras vs. PDF analítica")  plt.legend()  plt.grid()  plt.show() |