Termodinâmica Estatística I

Processos Markovianos e Teorema do Limite Central

Rafael Soares Andrade 96921
Humberto Sousa Martins 93724

Objetivo
O objetivo do projeto é, a partir de ferramentas computacionais, aplicar ideias de distribuições de probabilidades, processos Markovianos e da Teoria do Limite Central, obter dados de simulação construir gráficos e verificar aproximações da função Gaussiana.

Procedimento e Dados
O projeto foi desenvolvido na linguagem de programação Python versão 3.6, a partir da ferramenta de edição e compilação "Google Colaboratory". Para a construção dos gráficos, foi utilizada a ferramenta de plotagem Matplotlib. Foram usadas as bibliotecas numpy para manejo de matrizes e operações vetoriais e biblioteca scipy para ajuste de curvas também foram utilizadas.

A partir do roteiro, a primeira "tarefa"realizada foi obter a distribuição de probabilidade de uma variável que assume um valor aleatório entre 0 e 1. Para isso, foi utilizado uma ferramenta para geração de números aleatórios da biblioteca "numpy". Após escolher os parâmetros e a quantidade de números aleatórios gerados, podemos encontrar a distribuição de probabilidade dessa variável. No programa desenvolvido, esse processo é feito chamando uma função "histograma"que efetuará esse cálculo.

A função "histograma" funciona recebendo os dados gerados aleatoriamente, e o parâmetro utilizado para definir as subdivisões do gráfico. De acordo com os dados e parâmetros utilizados, função ajusta a escala gráfica e o limite dos compartimentos, denominados "bins". Após fazer os ajustes de escala e a normalização da função, o histograma é construído como mostrado na figura 1, e a função retorna os dados de frequência e coordenadas dos compartimentos.

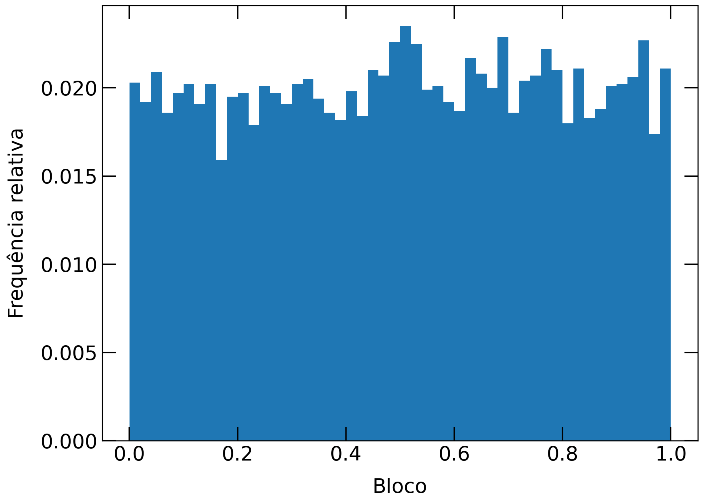
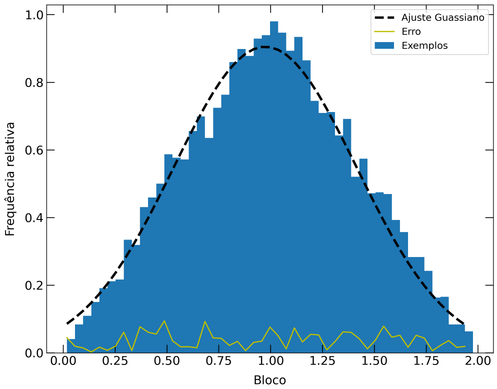
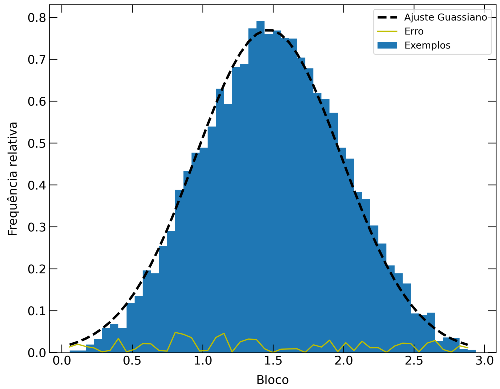


Figura 1: Histograma de frequência relativa de uma variável aleatória. O gráfico representa a frequência em que os números de 0 a 1 aparecem em uma geração aleatória dentro desse intervalo.

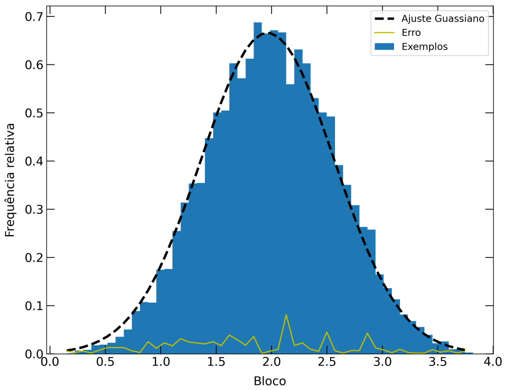
Após obter a distribuição de probabilidade de uma variável aleatória entre 0 e 1, obtivemos dados para as somas de duas ou mais variáveis aleatórias dentro do mesmo intervalo. Foi definida uma função que efetua o cálculo da distribuição de probabilidade ("Prob"). Tal parte do código recebe atributos: quantidade de variáveis aleatórias serão somadas(denominadas "n_termos"), parâmetros do número de compartimentos (denominado "n_bins"), e se há a necessidade de construir o gráfico ao ser chamada. A função gera "n_termos" números aleatórios e faz a soma desses números, adiciona o valor obtido a uma lista e repete esse processo "n_it" vezes. A partir dessa distribuição o histograma é construído, assim como um ajuste Gaussiano. Calculamos o erro entre a distribuição obtida e o ajuste gaussiano. Os resultados são mostrados nos gráficos a seguir.



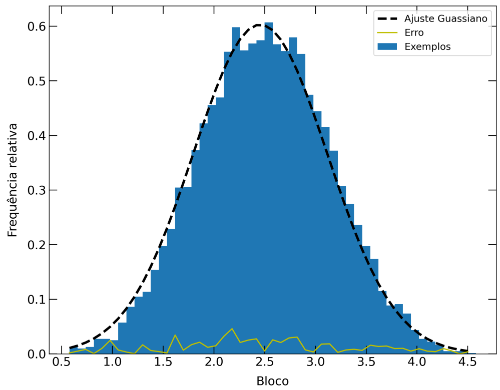
(a) Soma de 2 termos aleatórios



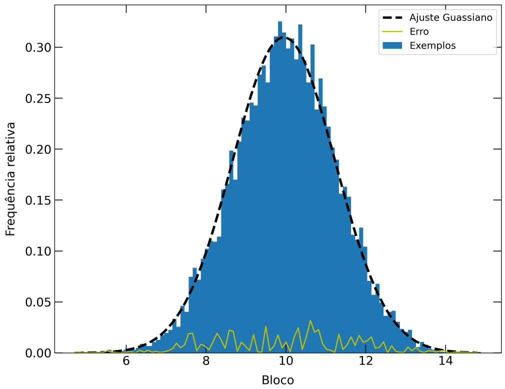
(b) Soma de 3 termos aleatórios



(c) Soma de 4 termos aleatórios



(d) Soma de 5 termos aleatórios



(e) Soma de 20 termos aleatórios

Figura 2: Função $P(x)$ para a distribuição de probabilidade da soma de variáveis aleatórias no intervalo entre 0 e 1. As barras azuis representam a frequência dos números gerados, a curva pontilhada representa o ajuste Gaussiana e a linha amarela representa o erro entre as duas distribuições.

Com os gráficos da função $P(x)$, podemos concluir que, quanto maior o número de variáveis somadas, mais a função $P(x)$ se aproximará de uma distribuição Gaussiana, diminuindo o seu erro. Foi desenvolvida uma função para testar tal informação, através de um gráfico. A função "erro"utiliza da mesma função "Prob"citada anteriormente, assumindo valores definidos previamente dos termos a serem somados. Para que não sejam construídos os mesmos gráficos já apresentados, a informação de que não há a necessidade dessa construção é enviada ("graph=False"), resultando em uma compilação com maior fluidez. Como a função "Prob" retorna o valor do erro, essa informação é armazenada em um "array"e utilizada para a construção do gráfico de erro por número de termos, mostrado na figura 3, que é mostrado a seguir

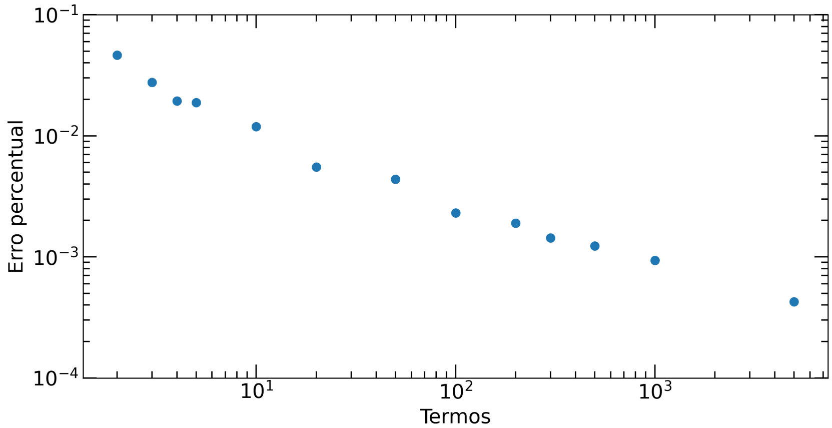


Figura 3: Gráfico que representa o erro entre a distribuição obtida e o ajuste Gaussiano, para parâmetros de vários termos. A curva decrescente mostra que, conforme a quantidade de termos somados aumenta, o erro da função em relação à distribuição Gaussiana diminui.

Para obter uma melhor visualização desta relação, extrapolamos o número de termos somados. Com essa construção foi concluído que, à medida em que aumentamos a quantidade de termos aleatórios somadas, menor será o erro. Somando 5000 termos, o erro se torna muito pequena, tendo a função $F(x)$ se aproximando da distribuição Gaussiana quando comparada à soma de menos termos.

O código fonte, assim como observações sobre sua execução seguem em anexo!

OBS: O código fonte está formatado para ser compilado pelo programa "Jupyter Notebook" ou o Google Colaboratory, e deve ser executado em Python nas versões 3.6 ou superior.

Recomendamos acesso ao código através do link:
https://colab.research.google.com/drive/1GMZlALHJDjcav1D4lt6xMygcpDV9casI?usp=sharing

---

```
# -*- coding: utf-8 -*-
"""ListaTermo.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
        GMZlALHJDjcav1D4lt6xMygcpDV9casI
"""

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MultipleLocator
from scipy.stats import norm
from scipy.optimize import curve_fit

rng = np.random.default_rng()

def histograma(dados, bins):
  '''Recebe uma lista de dados, plota um histograma e retora os
     valores do histograma'''

# estabelece os valores de minimo e m ximo e cria o espa o em
   x subdividido em bins (compartimentos)
  x_min = np.min(dados)
  x_max = np.max(dados)
  x = np.linspace(x_min,x_max,bins+1)
  interval = np.abs(x_max-x_min)
  y=[]

# faz a contagem dos valores de modo que o o valor ser  contado
    se estiver no intervalo do bin [x,x+1)
  for i in range(bins):
    counts = 0
    for j in dados:
      if j>=x[i] and j<x[i+1]:
        counts +=1

# caso algum dado tenha o mesmo valor do limite    direita do
```

```
      ltimo   bin , ele  ser   adicionado ao ultimo bin
      y.append(counts)
    for j in dados:
      if j == x_max:
       y[-1] +=1

# faz a normaliza  o do intervalo
    y=[i/len(dados) for i in y]

# plota o gr fico do hisograma
    plt.rcParams['legend.fontsize'] = 12
    plt.rcParams['font.size'] = 15
    plt.bar(x[0:bins],y,align='edge',width=interval/bins)
    plt.xlabel('Bloco',labelpad=10)
    plt.ylabel('Frequ ncia relativa',labelpad=10)
    plt.tick_params(direction='in',length=10, width = 1 , right =
        'on' , top = 'on')


# verifica erros
    if np.sum(y) < 0.9999999999999998:
      print('erro de normaliza  o' , np.sum(y))

#retorna o espa o x dos bins e y das frequ ncias
    return x,y

# estabelecemos alguns par metros
n_bins=50
n_exemplos=10000

# definimos uma seed (igual    matricula) e geramos um conjunto
    x de n meors aleat rios
np.random.seed(93724)
x = rng.uniform(0.,1.,n_exemplos)

# plotar e salvar o histograma
plt.figure(figsize=(8,6))
hist = histograma(x, n_bins)
plt.savefig('hist1',dpi=300)

def Prob(n_termos,n_it,n_bins,graph=True):

# Cria uma lista de valores vazia, realiza a soma de n_termos
    n meros aleat rios, n_it vezes e a cada itera  o adiciona
     o valor obtido    lista
    dist=[]
    for n in range(n_it):
      soma = np.sum(rng.uniform(size=n_termos))
      dist.append(soma)
```

```
# Aplica a fun  o histograma   lista de valores
  hist  = plt.hist(dist,bins=n_bins,density=True)
  plt.close()

# Faz o fit gaussiano, come amos com um chute da m dia e
    vari ncia da distribui  o , o fit em si   gerado pela
    gun  o curve_fit que retorna os valores ajustados da m dia
     e vari ncia
  mean = np.mean(dist)
  var = np.var(dist)
  x = np.array(hist[1][0:-1])
  y = np.array(hist[0])
  f_gaussiana = lambda x,mean,var: norm.pdf(x, mean,var)
  parametros_gaussiana = curve_fit(f_gaussiana,x,y,p0=[mean,var
      ])
  p = norm.pdf(x,parametros_gaussiana[0][0],(
      parametros_gaussiana[0][1]))

# Calcula o erro relativo
  erro = np.abs(hist[0] - p)

# Plota o gr fico
  if graph:
      plt.figure(figsize=(10,8))
      plt.tick_params(direction='in',which='minor',length=5,
          width = 1 , right = 'on' , top = 'on')
      plt.tick_params(direction='in',length=10, width = 1 , right
           = 'on' , top = 'on')
      plt.hist(dist,density=1,bins=n_bins,label=f"Exemplos")
      plt.plot(x, p, color='k',ls='--', linewidth=3,label='Ajuste
          Guassiano')
      plt.plot(x,erro,'y',label='Erro')
      plt.rcParams['legend.fontsize'] = 12
      plt.rcParams['font.size'] = 15
      plt.legend()
      plt.xlabel('Bloco',labelpad=10)
      plt.ylabel('Frequ ncia relativa',labelpad=10)
      plt.savefig(f'hist_{n_termos}_termos',dpi=300)
  return np.sum(erro)/n_bins

Prob(4,10000,50,graph=True)

def erro(n_it,n_bins):
  erro=[]
  termos=[2,3,4,5,10,20,50,100,200,300,500,1000,5000]

# Usamos a fun  o definida acima para calcular os erros
    relaivos com uma s rie de termos
```

```
    for i in termos:
        err=Prob(i, n_it, n_bins, graph=False)
        erro.append(err)

# Plotamos os resultados
    plt.figure(figsize=(10,5))
    plt.tick_params(direction='in',length=10, width = 1 , right =
        'on' , top = 'on')
    plt.tick_params(direction='in',which='minor',length=5, width =
        1 , right = 'on' , top = 'on')
    plt.scatter(termos,erro,label="erro")
    plt.yscale('log')
    plt.xscale('log')
    plt.ylim((1e-4,1e-1))
    plt.xlabel('Termos')
    plt.ylabel('Erro')
    plt.rcParams['legend.fontsize'] = 12
    plt.rcParams['font.size'] = 15
    plt.savefig('erro1',dpi=300)
    return

erro(10000,100)
```