

# MAC0110 — Terceiro EP - Data da Entrega: 1/5/2016

Roberto Hirata Jr.

21 de abril de 2016

## 1 Introdução

Neste terceiro EP, o objetivo será implementar, em Python, funções que calculam e apresentam alguns gráficos que complementarão o estudo feito no segundo EP.

Para completar este EP, será necessário o conhecimento de listas e tuplas, além de saber como utilizar a biblioteca `matplotlib` para gerar gráficos. Mas não se preocupe, parte será explicado em aula e parte estará explicado aqui neste texto. Em especial, na Seção 6.2, estão listados alguns links para a consulta desses tópicos.

## 2 Especificação das funções

As funções devem ter obrigatoriamente os nomes e os parâmetros abaixo especificados.

**Nota bene:** É importante deixar claro que você está livre para organizar o código de outras maneiras, isto é, pode criar outras funções que achar convenientes. Porém, o requisito mínimo é que as funções especificadas nas subseções abaixo sejam implementadas.

### 2.1 Transformações sobre listas uniformes

Nessa primeira parte, vamos criar algumas funções que fazem transformações sobre variáveis distribuídas uniformemente.

- `def transformacaoEscala(U, V, n, a, b):` A função deve receber duas listas, `U` e `V`, ambas de tamanho `n`, `U` com valores amostrados de uma distribuição uniforme, por exemplo, números entre 0 e 1, além de dois valores reais, `a` e `b`.

A sua função deve transformar a lista `V` de modo que, para uma posição genérica  $i$  de `V` o novo valor seja igual a:  $a + U[i] \cdot (b - a + 1)$ .

- `def somaVetores(U1, U2, U, n):` A função deve receber duas listas, `U1` e `U2`, ambas de tamanho `n`, com valores amostrados uniformemente. Além delas, uma lista `U` de mesmo tamanho, preenchida com zeros (pode ser qualquer número, na verdade).

A sua função deve transformar a lista  $U$  de modo que, para uma posição genérica  $i$  de  $U$ , o novo valor seja igual a:  $U1[i] + U2[i]$ .

- **def raizVetor(U,n):** A função deve receber uma lista,  $U$ , de tamanho  $n$ ,  $U$  com valores amostrados uniformemente, como anteriormente.

A sua função deve transformar a lista  $U$  de modo que, para uma posição genérica  $i$  de  $U$ , o novo valor seja igual a:  $\sqrt{U[i]}$ .

- **def normalTransform(U1, U2, Z1, Z2, n):** A função deve receber duas listas,  $U1$  e  $U2$ , ambas de tamanho  $n$ , com valores amostrados uniformemente no intervalo  $(0, 1)$ . Além delas, duas listas,  $Z1$  e  $Z2$  de mesmo tamanho das primeiras, preenchida com zeros.

A sua função deve transformar  $Z1$  e  $Z2$  de forma que, para uma posição genérica  $i$ , elas sejam iguais a:

$$Z0[i] = \cos(2\pi U2[i]) * \sqrt{-2 \ln U1[i]}$$

$$Z1[i] = \sin(2\pi U2[i]) * \sqrt{-2 \ln U1[i]}$$

**Obs.:** A transformação acima é conhecida como Box-Muller. Para quem tiver curiosidade, veja a página da Wikipedia: [Wikipedia - Box-Muller Transform](#).

- **def histogram(U,n,H,m):** A função deve receber duas listas,  $U$  e  $H$ , com tamanhos  $n$  e  $m$ , respectivamente. A lista  $U$  é preenchida com valores amostrados uniformemente e a lista  $H$  pode estar preenchida com qualquer valor, isto é, você deve zerar essa lista antes de qualquer coisa na sua função.

A função deve calcular e preencher o histograma dos valores de  $U$  em  $m$  intervalos.

## 2.2 Programa principal

Para testar as funções, faça um programa que:

1. leia um número inteiro positivo  $n$  e dois valores reais  $a$  e  $b$ , gere um histograma da lista resultante da função `transformacaoEscala`, grafique o histograma e salve a imagem na pasta de execução do programa com o nome `hist_escala_translacao.png`;
2. grafique o histograma da lista resultante da função `somaVetores` e salve a imagem na pasta de execução do programa com o nome `hist_soma.png`.
3. grafique o histograma da lista resultante da função `raizVetor` e salve a imagem na pasta de execução do programa com o nome `hist_raiz.png`.
4. grafique o histograma da lista resultante da função `normalTransform` para as listas  $Z0$  e  $Z1$  e salve a imagem na pasta de execução do programa com os nomes `hist_normal_Z0.png` e `hist_normal_Z1.png`, respectivamente.

## 2.3 Análise da distribuição das cordas geradas no EP2

No EP2, vimos três métodos que se diferenciavam na forma de criação das cordas.

Adapte seu EP2 criando as seguintes funções:

- Lista de cordas do método 1 - `def listaCordasM1(r,n).`
- Lista de cordas do método 2 - `def listaCordasM2(r,n).`
- Lista de cordas do método 3 - `def listaCordasM3(r,n).`

Essas funções devem receber o tamanho do raio ( $r$ ), o número de cordas ( $n$ ) a serem geradas e devem retornar uma lista de tuplas que contém as coordenadas dos pontos extremos das  $n$  cordas. Note que, aqui, você deve criar a listas de pares de coordenadas dentro de sua função e retorná-la com `return`.

As cordas devem ser geradas exatamente da forma esperada pelos métodos. No entanto, você é livre para escolher o sistema de coordenadas para representar os pontos extremos das cordas dessas cordas, ou seja, elas podem ser representadas em coordenadas polares,  $((\rho_A, \theta_A), (\rho_B, \theta_B))$ , ou em coordenadas cartesianas,  $((x_A, y_A), (x_B, y_B))$ .

Para auxiliar a implementação das funções acima e das demais funções do EP, crie as seguintes funções:

1. Determina os pontos médios das cordas dados seus extremos -

`def determinaPontosMedios(listaExtremos):`

Essa função deve receber uma lista de tuplas contendo as coordenadas dos pontos extremos das  $n$  cordas. A função deve devolver uma lista de tuplas contendo as coordenadas dos pontos médios das  $n$  cordas.

Atenção para preservar a ordem com que foram listadas as cordas.

2. Determina os pontos extremos das cordas dado o seu ponto médio -

`def determinaPontosExtremos(r,listaPontosMedios):`

Essa função deve receber o tamanho ( $r$ ) do raio e uma lista de tuplas contendo as coordenadas dos pontos médios das  $n$  cordas. A função deve devolver uma lista de tuplas contendo as coordenadas dos pontos extremos das  $n$  cordas.

Atenção para preservar a ordem com que foram listadas as cordas.

Finalmente, agora vocês devem criar as funções principais do EP:

1. Análise da distribuição dos pontos extremos das cordas na circunferência (borda do círculo) -

`def distribuicaoBorda(r,listaCordas):`

Essa função deve receber o tamanho ( $r$ ) do raio e uma lista de tuplas contendo as coordenadas dos pontos extremos das  $n$  cordas.

Utilizando esses pontos, sua função deve:

- Fazer uma partição da borda do círculo em 8 arcos de mesmo comprimento;
- Categorizar os pontos extremos das cordas de acordo com o arco ao qual o ponto pertence. Para isso, pode-se criar uma lista que contém os nomes das categorias dos  $2n$  pontos extremos;
- Em um gráfico, esboçar o círculo de raio  $r$  e os  $2n$  pontos extremos categorizados por cor;
- Ao lado, na mesma imagem, esboçar o histograma que mostra a frequência (quantidade de pontos pertencentes) de cada categoria;
- Salvar a imagem em formato png na mesma pasta de execução do programa com o nome "distribuicao\_borda.png".

2. Análise da distribuição dos pontos médios das cordas no sentido radial -

`def distribuicaoRadial(r, listaCordas):`

Essa função deve receber o tamanho ( $r$ ) do raio e uma lista de tuplas contendo as coordenadas dos pontos extremos das  $n$  cordas.

Utilizando esses pontos, sua função deve:

- Determinar a lista dos  $n$  pontos médios de cada corda a partir de seus pontos extremos;
- Fazer uma partição do círculo em 8 coroas circulares (aneis) de mesma largura;
- Categorizar os pontos médios das cordas de acordo com a coroa a qual o ponto pertence. Para isso, pode-se criar uma lista que contém os nomes das categorias dos  $n$  pontos médios;
- Em um gráfico, esboçar o círculo de raio  $r$  e os  $n$  pontos médios categorizados por cor;
- Ao lado, na mesma imagem, esboçar o histograma que mostra a frequência (quantidade de pontos pertencentes) de cada categoria;
- Salvar a imagem em formato png na mesma pasta de execução do programa com o nome "distribuicaoRadial.png".

3. Análise da distribuição dos pontos médios das cordas em área -

`def distribuicaoArea(r, listaCordas):`

Essa função deve receber o tamanho ( $r$ ) do raio e uma lista de tuplas contendo as coordenadas dos pontos extremos das  $n$  cordas.

Utilizando esses pontos, sua função deve:

- Determinar a lista dos  $n$  pontos médios de cada corda a partir de seus pontos extremos;

- Fazer uma partição do círculo em 8 regiões de mesma área; Aqui, tome o cuidado de **não** escolher a divisão em 8 setores iguais, senão observaremos a distribuição radial, não por área.
- Categorizar os pontos médios das cordas de acordo com a região a qual o ponto pertence. Para isso, pode-se criar uma lista que contém os nomes das categorias dos  $n$  pontos médios;
- Em um gráfico, esboçar o círculo de raio  $r$  e os  $n$  pontos médios categorizados por cor;
- Ao lado, na mesma imagem, esboçar o histograma que mostra a frequência (quantidade de pontos pertencentes) de cada categoria;
- Salvar a imagem em formato png na mesma pasta de execução do programa com o nome "distribuicao\_area.png".

#### 4. Análise da distribuição das cordas -

```
def distribuicaoCordas(r, listaCordas):
```

Essa função deve receber o tamanho ( $r$ ) do raio e uma lista de tuplas contendo as coordenadas dos pontos extremos das  $n$  cordas.

Utilizando esses pontos, sua função deve:

- Esboçar em um gráfico o círculo de raio  $r$  e traçar as  $n$  cordas dados os seus pontos extremos;
- Salvar a imagem em formato png na mesma pasta de execução do programa com o nome "distribuicao\_cordas.png".

Escreva uma função principal (**main**) que leia os parâmetros  $r$  e  $n$  do teclado e, para as listas de cordas geradas em cada um dos três métodos vistos no EP2, execute as funções `distribuicaoBorda`, `distribuicaoRadial`, `distribuicaoArea` e `distribuicaoCordas`.

## 3 Relatório

Analise o efeito causado pelas transformações implementadas na seção anterior, quando aplicadas em listas de valores amostrados de uma distribuição uniforme. Para isso, experimente as funções e analise os gráficos gerados.

Adicionalmente, analise, para as listas de cordas geradas em cada um dos três métodos, a distribuição dos pontos extremos na borda, a distribuição dos pontos médios tanto de forma radial como por área, além do aspecto observado no gráfico com as cordas traçadas.

Crie um relatório em formato **notebook** do Python onde são mostradas todas as imagens dos gráficos gerados no EP, além de fazer uma breve análise sobre as distribuições observadas. Comente a relação dessas distribuições observadas com as probabilidades encontradas no EP2.

## 4 Entrega

A entrega consiste de um arquivo comprimido em formato zip com os arquivos .py e o arquivo .ipynb. Não inclua as imagens para o arquivo não ficar gigantesco.

## 5 Plágio

Plágio é a cópia/modificação não autorizada e/ou sem o conhecimento do autor original. O plágio é um problema grave que pode levar até a expulsão do aluno da universidade. Para quaisquer dúvidas, consulte o texto que disponibilizamos para uma disciplina irmã desta (Plágio).

## 6 Alguma ajuda

### 6.1 Tutorial básico

Abaixo segue um tutorial específico para os gráficos que estão sendo pedidos neste EP:

```
import matplotlib.pyplot as plt
import numpy as np

# especifica o tamanho da imagem e a resolucao
plt.figure(figsize=(15, 6), dpi=80)

# trabalhando no primeiro gráfico (lado esquerdo da figura)
plt.subplot(121)

# desenhando um círculo de raio r = 10
thetas=np.arange(0,2*np.pi,0.01)
r=10
plt.plot(r*np.cos(thetas), r*np.sin(thetas), 'b-')

# adicionando alguns pontos e retas dentro do círculo
plt.plot([1,2,3,4], [5,6,7,8], 'yo')
plt.plot([1,2,3,4], [5,6,7,8], 'g-')
plt.plot([-2.5,-3,-4.5], [-5,-6.5,-7.5], 'go')
plt.plot([-2.5,-3,-4.5], [-5,-6.5,-7.5], 'r-')

# customizando o gráfico
plt.ylabel('Gráfico 1 - Nome do eixo Y')
```

```

plt.xlabel('Gráfico 1 - Nome do eixo X')

#Especifica os intervalos dos eixos com [xmin, xmax, ymin, ymax]
plt.axis([-r, r, -r, r])

plt.title('Título do Gráfico 1')

# trabalhando no segundo gráfico (lado direito da figura)
plt.subplot(122)
x = [1,2,3,4,5,6,7,8,1,1,1,2,2,3,3,3,3,3,5,5,5,5,6,6,7,7,7,8,8,8,8,8]
plt.hist(x, 8, facecolor='g')
plt.title('Histograma')

plt.ylabel('Gráfico 2 - Nome do eixo Y')
plt.xlabel('Gráfico 2 - Nome do eixo X')

# comente a linha abaixo caso queira salvar o gráfico
plt.show()

# descomente a linha abaixo se deseja salvar o gráfico
# plt.savefig('example.png')

```

## 6.2 Referências

Alguns links que podem ser úteis:

- **Especificação do pyplot:** API do pyplot.
- **Tutorial 1:** Tutorial do pyplot.
- **Tutorial 2:** Scipy lectures.
- **FAQ do matplotlib:** How to e FAQ do matplotlib.
- **Video tutorial:** Para um tutorial mais completo e mais detalhado, veja o vídeo do Eric Jones: YouTube - Tutorial de Numpy e Matplotlib
- **Listas e tuplas:** Python 3.3 docs: Sequences and Tuples
- **Transformação Box-Muller:** Wikipedia: Box-Muller Transform.