Algoritmos Genéticos para otimização de funções em Python

Introdução.- Os algoritmos genéticos (AG) são baseados no processo de seleção natural e podem ser utilizados como um método para resolução de problemas de otimização. O algoritmo genético modifica repetitivamente a população de indivíduos (geração) que representam as soluções do problema. A cada iteração, o algoritmo genético seleciona de forma aleatória ou utilizando alguma heurística, alguns indivíduos que serão utilizados como pais para gerar filhos para a seguinte geração. Após sucessivas gerações, a população evolui convergindo para uma solução ótima.
Os algoritmos genéticos, entre outras aplicações, podem ser utilizados para solucionar problemas de otimização; inclusive para otimizar aqueles problemas em que os algoritmos comuns são ineficazes. Eles são capazes de otimizar inclusive problemas onde a função objetivo é descontínua não diferençável, estocástica ou não linear.
Os AG utilizam três tipos de regras a cada iteração para criar a próxima geração a partir da população atual:

- **Regras de seleção** de indivíduos da população, chamados pais, que contribuirão para gerar a próxima geração.
- **Regras de Crossover** (Cruzamento) para combinar os genes de dois indivíduos pais que irão gerar filhos para a próxima geração.
- **Regras de mutação** que aplicam mudanças aleatórias nos genes dos indivíduos pais

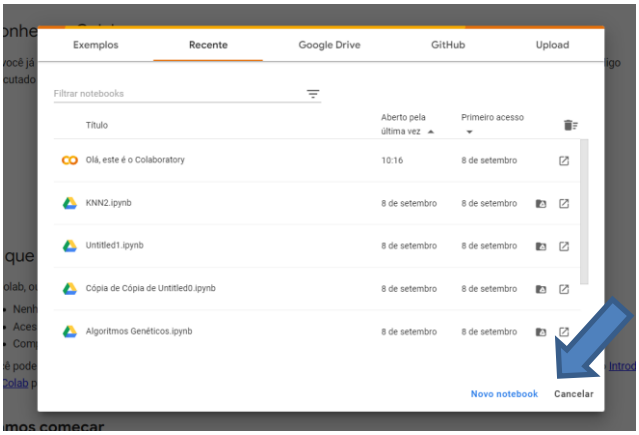
Exercicio 1.

Considere o seguinte caso exemplo. Deseja-se minimizar a seguinte função:

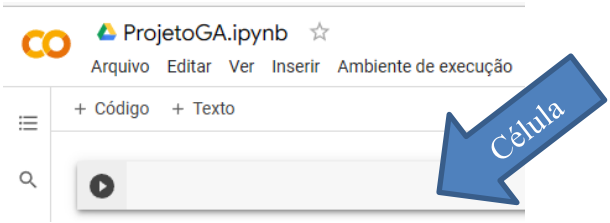
$$\boldsymbol{f(t) = e^{-t} \times \cos(2\pi t)}$$

Para esta prática iremos utilizar o ambiente Colab Research. Este ambiente possibilita o desenvolvimento de software na linguajem Python utilizando um navegador web.

1. Acesse o ambiente Colab através do link: https://colab.research.google.com
2. Faça login utilizando seu e-mail e senha da USP
3. Na tela inicial clique em criar **Novo notebook**, este procedimento irá criar um notebook Jupyter. (Para mais detalhes sobre o Notebook Jupyter, visite o site www.jupyter.org)



4. Renomeie o notebook para ProjetoGA, clicando sobre o nome atua



5. Instale a biblioteca “geneticalgorithm”. Clique na célula do notebook e digite o seguinte código que serve para instalar essa biblioteca externa, depois clique no ícone ‘Executar Célula’ 

```
!pip install geneticalgorithm
```

“genetcalgorithm” *é uma biblioteca desenvolvida para facilitar a implementação e emprego dos algoritmos genéticos na linguagem python, para solução de problemas de otimização e busca.*

6. Crie uma nova célula de código clicando em “+ Código”. Digite e execute o seguinte código que chama as bibliotecas que serão utilizadas no programa Python:

```
import numpy as np
import matplotlib.pyplot as plt
from geneticalgorithm import geneticalgorithm as ga
```

7. Em uma nova célula, digite e execute o código a seguir que cria a função $\boldsymbol{f(t)}$ a ser minimizada (chamada de função de aptidão ou fitness).

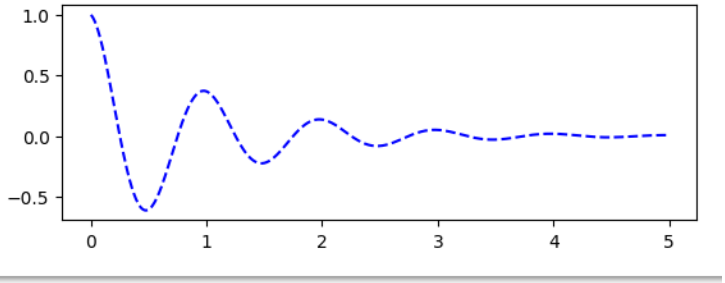
```
def f(t):
    z=np.exp(-t) * np.cos(2*np.pi*t)
    return z
```

$$\boldsymbol{f(t) = e^{-t} \times \cos(2\pi t)}$$

“np” *é o objeto baseado na biblioteca* ***numpy*** *que é utilizada para realizar operações numéricas e matemáticas, fornece suporte para matrizes e funções para manipulação de dados numéricos.*

8. O seguinte código pode ser utilizado para plotar a função $f(t)$:

```
t=np.arange(0,5,0.01) #Gera valores no intervalo 0 a 5 com passo de 0.01
plt.figure(1)
plt.subplot(212)
plt.plot(t,f(t),'b--')
plt.show()
```



*"**plt**" é o objeto baseado na biblioteca matplotlib utilizada para a criação de gráficos e visualizações de dados.*

9. O seguinte código cria dicionário* que contêm os parâmetros que serão utilizados no algoritmo genético:

```
#Parâmetros do algoritmo genético
algorithm_param = {'max_num_iteration': 1000,\
                   'population_size':100,\
                   'mutation_probability':0.1,\
                   'elit_ratio': 0.01,\
                   'crossover_probability': 0.5,\
                   'parents_portion': 0.3,\
                   'crossover_type':'uniform',\
                   'max_iteration_without_improv':None}
```

** Os dicionários python são coleções de itens compostos por uma chave e um valor.*

***max_num_iteration:** É o critério de parada do algoritmo. Se o valor deste parâmetro for "None", o algoritmo define automaticamente o número máximo de iterações como uma função da dimensão, limites e tamanho da população.*

***population_size:** determina o número de soluções de teste em cada iteração. O valor padrão é 100.*

***mutation_probability:** determina a probabilidade de cada gene sofrer mutação aleatória em cada solução individual. O padrão é 0,1 (ou seja, 10 por cento).*

***elit_ratio:** determina o número de elites na população. O valor padrão é 0,01 (ou seja, 1 por cento). Por exemplo, quando o tamanho da população é 100 e elit_ratio é 0,01, então há uma elite na população. Se este parâmetro for definido como zero, o geneticalgorithm implementa um algoritmo genético padrão em vez de um algoritmo genético elitista.*

***crossover_probability:** determina a probabilidade de uma solução existente passar seu genoma para seus descendentes; o valor padrão é 0,5 (ou seja, 50 por cento).*

***parents_portion:** a porção da população preenchida pelos membros da geração anterior; o padrão é 0,3 (ou seja, 30 por cento da população).*

***crossover_type:** existem três opções: one_point (ponto único), two_point (dois pontos) e funções de crossover uniforme; o padrão é o crossover uniforme.*

***max_iteration_without_improv:** se o algoritmo não melhorar a função objetivo ao longo do número de iterações sucessivas determinadas por este parâmetro, então o geneticalgorithm para e reporta a melhor solução encontrada antes de atingir o max_num_iterations. O valor padrão é None.*

10. Crie um vetor que representa o intervalo da solução da função $f(t)$ no algoritmo genético:

```
[12] intervalo=np.array([[0,5]]) # será utilizado para definir o intervalo da solução
```

11. O código a seguir, cria um modelo de algoritmo genético baseado na biblioteca "genetcalgorithm". Ao executar a função "run" o algoritmo inicia e o ciclo de evolução é realizado até alcançar o número máximo de iterações definidas no parâmetro "max_num_iteraction".

```
model=ga(function=f,dimension=1,variable_type='real',variable_boundaries=intervalo,algorithm_parameters=algorithm_param)
model.run()
```

```
...  |||||||||_____________________________________ 20.0% GA is running...
```

Onde:

Function *é a função a ser minimizada.*

Dimension *é o número de variáveis de entrada da função.*

Variable_type *é o tipo de formatação dos dados de entrada da função ('bool', 'int' ou 'real').*

Variable_boundaries *é o intervalo da solução para cada variável.*

Algoritm_parameters *são os parâmetros do algoritmo genético.*

12. Após o término das iterações do algoritmo, a solução encontrada pode ser exibida através do seguinte comando:

```
solution=model.output_dict
print(solution)
```

```
{'variable': array([0.47488039]), 'function': -0.6142287973108481}
```

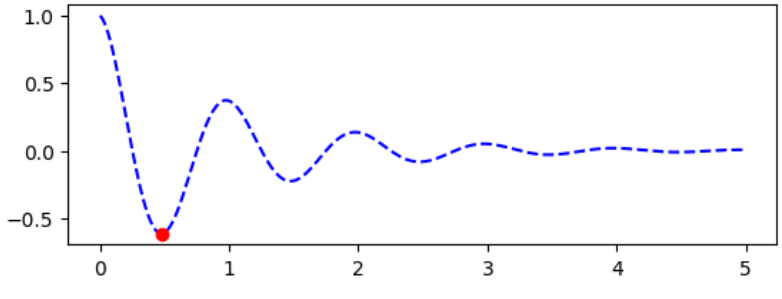
A execução resulta em um dicionário contendo duas chaves:

Variable *cujo valor é uma matriz que contém cada uma dos valores de entrada para os quais a saída da função é mínima (no intervalo definido).*

Function *cujo valor é a saída da função no ponto mínimo encontrado.*

13. O seguinte trecho de código pode ser utilizado para plotar o ponto mínimo encontrado para a função no intervalo definido.

```
x=solution['variable'][0] # O indice 0 é utilizado para a primeira variável de entrada
y=solution['function']
plt.figure(1)
plt.subplot(212)
plt.plot(t,f(t),'b--') # Plota a função
plt.plot(x,y,'ro') # Plota o ponto mínimo
plt.show()
```

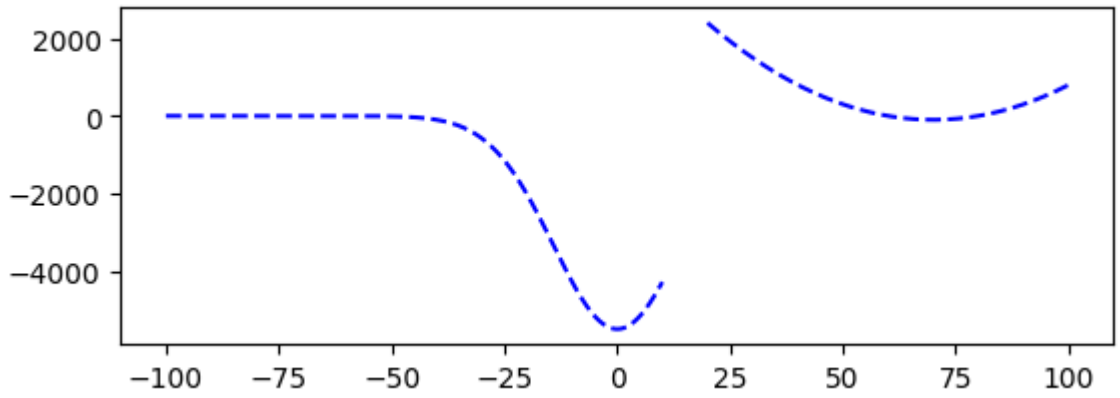


Exercicio 2.

Modifique o programa anterior para minimizar a seguinte função descontínua no intervalo de]10 , 20]:

$$f(x) = \begin{cases} -e^{-\left(\frac{x}{20}\right)^2} \times 5500, & x \leq 10 \\ -e^{-1} + (x-60)(x-80), & x > 20 \end{cases}$$

```python
def f(x):
    z = np.zeros(len(x)) #cria um vetor z com o número de elementos de x
    for i in range(len(x)): #para cada elemento do vetor x
        #Primeiro parâmetro
        if x[i] <= 10:
            z[i] = -np.exp(-(x[i]/20)**2) * 5500
        #Segundo parâmetro
        elif x[i] > 20:
            z[i] = -np.exp(-1) + (x[i] - 60) * (x[i] - 80)
        #Intervalo onde não existe imagem da função
        elif x[i] >10 and x[i] <=20:
            z[i]=np.nan #nan (not a number) insere um valor nulo
    return z
```



Obs. Utilize o intervalo de -100 a 100