# TAREFA 6 - Algoritmo Genético


- Diego Santos Gonçalves - 20183012537
- Mariana Bulgarelli Alves dos Santos - 20183000330


Para executar este notebook, é necessária a instalação das seguintes bibliotecas:

- numpy
- pandas
- random
- matplotlib - para criação dos gráficos
- mpl_toolkits - para criação dos gráficos 3d
- tqdm - para barra progresso

## Importe das bibiliotecas utilizadas no código

In [17]:

```
from tqdm import tqdm
import math
import random
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d
```

## Função Objetivo

Nós implementamos a função objetivo dada para o problema: função alpine 02. Neste caso nossa função terá apenas duas entradas (x1, x2).

In [18]:

```
# Funcao Objetivo
def f_alpine02(x):
    resultado = 1
    for i, x_i in x.iteritems():
        resultado = resultado * math.sin(x_i) * math.sqrt(x_i)
    return resultado
```

Com base na função objetivo, nós ordenamos a população que temos

In [19]:

```
# Ordenar
def ordena(df):
    df['aux'] = df.apply(f_alpine02, axis=1)
    df = df.sort_values('aux', ascending=False).drop('aux', axis = 1)
    return df.reset_index(drop=True)
```

## Função Fitness

A função de aptidão escolhida, foi o Ranking Linear. Nesta função é necessário ordenar pelo valor da função objetivos os individuos da população. Depois disso é atribuido um valor entre um máximo e um mínimo escolhido.

In [20]:

```
def ranking_linear(df, max, min):
    Fx = []
    df_ordenado = ordena(df)
    df_ordenado = df_ordenado.reset_index(drop = True)
    for i, row in df_ordenado.iterrows():
        resultado = min + (max - min)*(( len(df_ordenado) - i - 1)/( len(df_ordenado) -1))
        Fx.append(resultado)
    return Fx
```

## Mutação Uniforme

A mutação escolhida foi a uniforme. Neste caso, se para o atributo do individuo, se o valor randômico for menor que a taxa de mutação (5%) será atribuído um valor randômico do espaço de busca (entre 0 e 10).

In [21]:

```
# Mutacao
def mutacao_uniforme(individuo, tx_mutacao):
    individuo_mutado = []
    # como a taxa de mutacao eh entre 0 e 1, temos:
    for x_i in individuo:
        rand = np.random.random()
        if rand < tx_mutacao:
            x_i = np.random.uniform(0.0, 10.0)
        individuo_mutado.append(x_i)
    return individuo_mutado
```

## Crossover Aritmético

O crossover escolhido foi o aritmético. Aqui, um valor randomico alfa é selecionado. Esse alfa será utilizado para fazer combinação linear dos atributos dos pais para a geração dos filhos, caso seja menor que a taxa de crosover.

In [22]:

```
def crossover(P1, P2, tx_crossover):
    C1 = P1
    C2 = P2
    alpha = np.random.random()
    rand = np.random.random()
    for i in range(len(P1)):
        t = alpha*P1[i] + (1 - alpha)*P2[i]
        s = alpha*P2[i] + (1 - alpha)*P1[i]

        if (t >= 0 and t <= 10 and s >= 0 and s <= 10) and tx_crossover > rand:
            C1[i] = t
            C2[i] = s

    return (C1, C2)
```

## Metodo de Selecao : Roleta

Usando o Ranking Linear para fazer a soma acumulada, o indivíduo é escolhido com base num valor randômico (no caso utilizamos a função `random.triangular()` , que aumenta a probabilidade do valor escolhido estar próximo de um ponto escolhido, dessa forma queremos priorizar a escolha de melhores indivíduos).

In [44]:

```
# Metodo de selecao - roleta
def selecao_roleta(Fx, df):
    posicao = 0
    soma_acumulada = np.cumsum(Fx)
    tamanho = len(Fx)
    limite = soma_acumulada[tamanho-1]
    divisao = np.random.randint(1,9)
    rand = random.triangular(0, 100,limite)

    for i, valor in enumerate(soma_acumulada):
        if rand <= valor:
            posicao = i
            break

    return df.loc[posicao]
```

In [45]:

```
def plot_geracao_3d(df, geracao):
    populacao = df.copy()
    populacao['fx'] = populacao.apply(f_alpine02, axis=1)
    if geracao%10 == 0:
        print("Geracao: ", geracao)
        fig = plt.figure(figsize= (16,9))
        ax = plt.axes(projection = '3d')

        ax.grid(b = True, color = 'grey', linestyle = '-.', linewidth = 0.3, alpha = 0.2)

        my_cmap = plt.get_cmap('hsv')

        sctt = ax.scatter3D(populacao['X1'],populacao['X2'], populacao["fx"], alpha = 0.8, c=populacao["fx"], cma
p= my_cmap, marker = '.')

        plt.title(f"Funcao alpine2 - Geracao {geracao}")
        ax.set_xlabel('x1')
        ax.set_ylabel('x2')
        ax.set_zlabel('f(x1,x2)')
        fig.colorbar(sctt, ax=ax, shrink= 0.5, aspect = 5)

        plt.show()
        plt.clf()
        plt.close()
```

In [46]:

```
def plot_evolucao_temporal(melhores, piores, medias):
    x = [i for i in range(0,len(melhores))]

    y_melhor = []
    y_pior = []
    y_media = []

    for i in range(len(melhores)):
        y_melhor.append(f_alpine02(melhores[i]))
        y_media.append(f_alpine02(medias[i]))
        y_pior.append(f_alpine02(piores[i]))

    fig = plt.figure(figsize= (16,9))
    ax = plt.axes()


    plt.plot(x,y_melhor,'b-.', label = "Melhor")
    plt.plot(x,y_media, 'g-.', label = "Media")
    plt.plot(x,y_pior, 'r-.', label = "Pior")

    plt.title("Evolucao temporal do algoritmo")
    ax.set_xlabel('Geracao')
    ax.set_ylabel('f(x1,x2)')

    plt.legend()
    plt.show()
    plt.clf()
    plt.close()
```

In [47]:

```
def plot_variaveis_populacao(melhores, piores, medias):


    x1_melhores = []
    x2_melhores = []

    x1_medias = []
    x2_medias = []

    x1_piores = []
    x2_piores = []

    for i in range(len(melhores)):
        x1_melhores.append(melhores[i]['X1'])
        x2_melhores.append(melhores[i]['X2'])

        x1_medias.append(medias[i]['X1'])
        x2_medias.append(medias[i]['X2'])

        x1_piores.append(piores[i]['X1'])
        x2_piores.append(piores[i]['X2'])

    fig = plt.figure(figsize= (16,9))
    ax = plt.axes()


    plt.plot(x1_melhores,x2_melhores,'b-.', label = "Melhor")
    plt.plot(x1_medias,x2_medias, 'g-.', label = "Media")
    plt.plot(x1_piores,x2_piores, 'r-.', label = "Pior")

    plt.title("Evolucao dos atributos da populacao")
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')

    plt.legend()
    plt.show()
    plt.clf()
    plt.close()
```

In [48]:

```
def populacao_media(populacao):
    return populacao.mean(axis = 0)
```

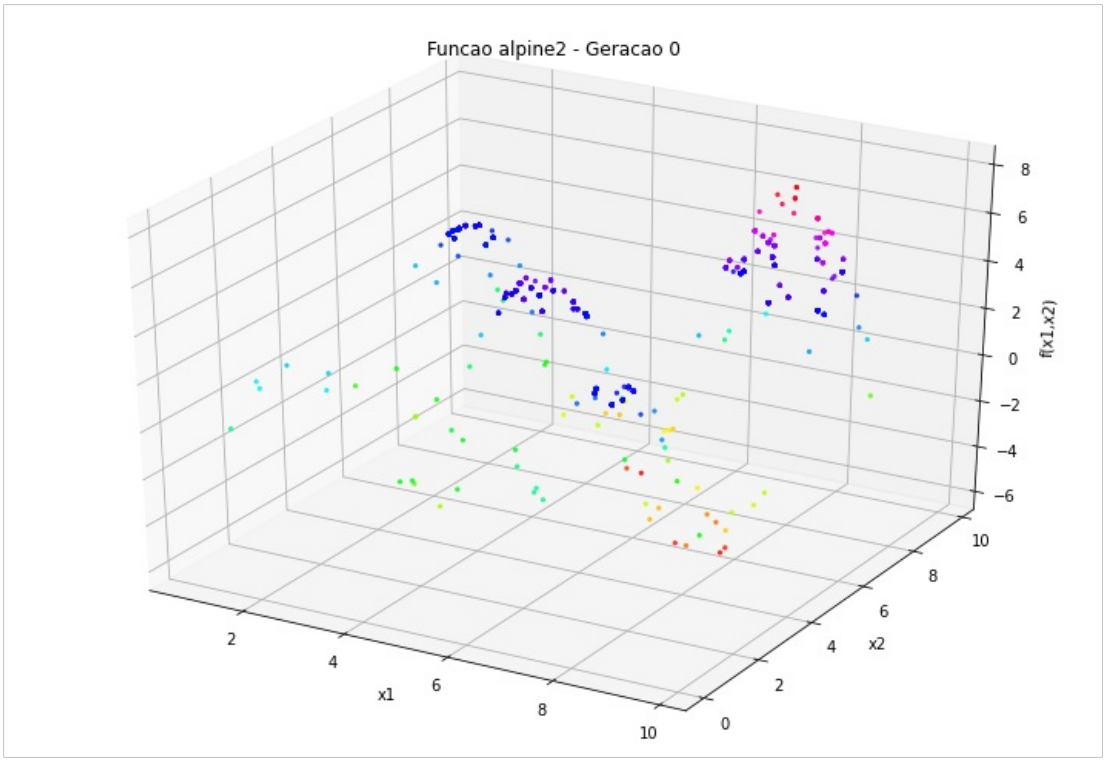
## Algorítmo Genético

Na célula abaixo está a implementação do algorítmo genético. Criamos uma população inicial com tamanho 1000 com atributos com valores aleatórios entre 0 e 10. Escolhemos uma taxa de mutação de 5% e uma taxa de cruzamento de 80%, que será aplicado nos métodos correspondentes. Após experimentos, vimos que 50 gerações já são suficiente para a convergência do método. Outra decisão que tivemos, era que um indivíduo não poderia se reproduzir com ele mesmo, de forma que não gere indivíduos duplicados.

In [49]:

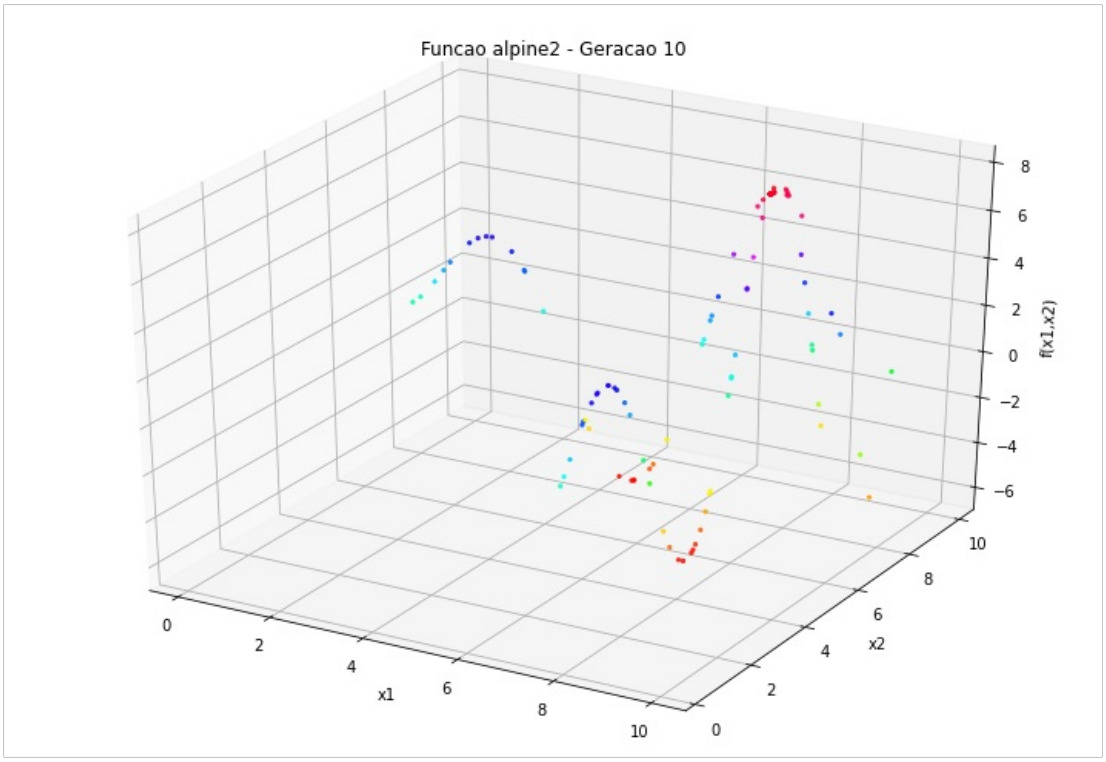
```
# Parametros do algoritmo genetico
tamanho_pop: int = 1000
tx_mutacao = 0.05
tx_crossover = 0.8
# num geracoes
total_geracoes = 50
melhor_geracao = 0

# Cria dataframe e preenche com valores randomicos
populacao = pd.DataFrame(np.random.uniform(0.0, 10.0, size=(tamanho_pop, 2)), columns=['X1', 'X2'])

#print(populacao)

# melhor
best = []
pior = []
melhor = []
media = []


for geracao in tqdm(range(total_geracoes)) :
    # ordena
    populacao_ordenada = ordena(populacao.copy()).reset_index(drop = True)
    # Pegar melhor individuo da geracao atual
    if geracao == 0:
        best = populacao_ordenada.loc[0]
    elif f_alpine02(best) < f_alpine02(populacao_ordenada.loc[0]):
        best = populacao_ordenada.loc[0]
        melhor_geracao = geracao

    melhor.append(populacao_ordenada.loc[0])
    pior.append(populacao_ordenada.loc[tamanho_pop-1])
    media.append(populacao_media(populacao_ordenada.copy()))

    Fx = ranking_linear(populacao_ordenada, 10, 0)
    populacao_ordenada = populacao_ordenada.drop('aux', axis=1)

    nova_populacao = pd.DataFrame(columns = ['X1', 'X2'])
    for i in range(0, int(tamanho_pop/2)):
        # chama selecao
        P1 = selecao_roleta(Fx, populacao_ordenada)
        # para que não haja repetição de individuos
        while(True):
            P2 = selecao_roleta(Fx, populacao_ordenada)

            if( P2['X1'] == P1['X1'] and P2['X2'] == P1['X2']):
                break

        # transforma P1 e P2 em vetor
        P1 = P1.to_numpy()
        P2 = P2.to_numpy()

        # faz crossover
        C1, C2 = crossover(P1, P2, tx_crossover)

        # Realizar mutacao de C1 e C2
        C1 = mutacao_uniforme(C1, tx_mutacao)
        C2 = mutacao_uniforme(C2, tx_mutacao)

        nova_populacao=nova_populacao.append(pd.DataFrame(data=[C1, C2], columns = ['X1', 'X2'] ))

    populacao = nova_populacao.reset_index(drop=True)
    plot_geracao_3d(populacao,geracao)

plot_geracao_3d(populacao, geracao)


print("========================================")
print("Best individuo: ")
print(best)
print("Funcao alpine02: ",f_alpine02(best))
print("Geracao: ",melhor_geracao)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]

Geracao:  0
```



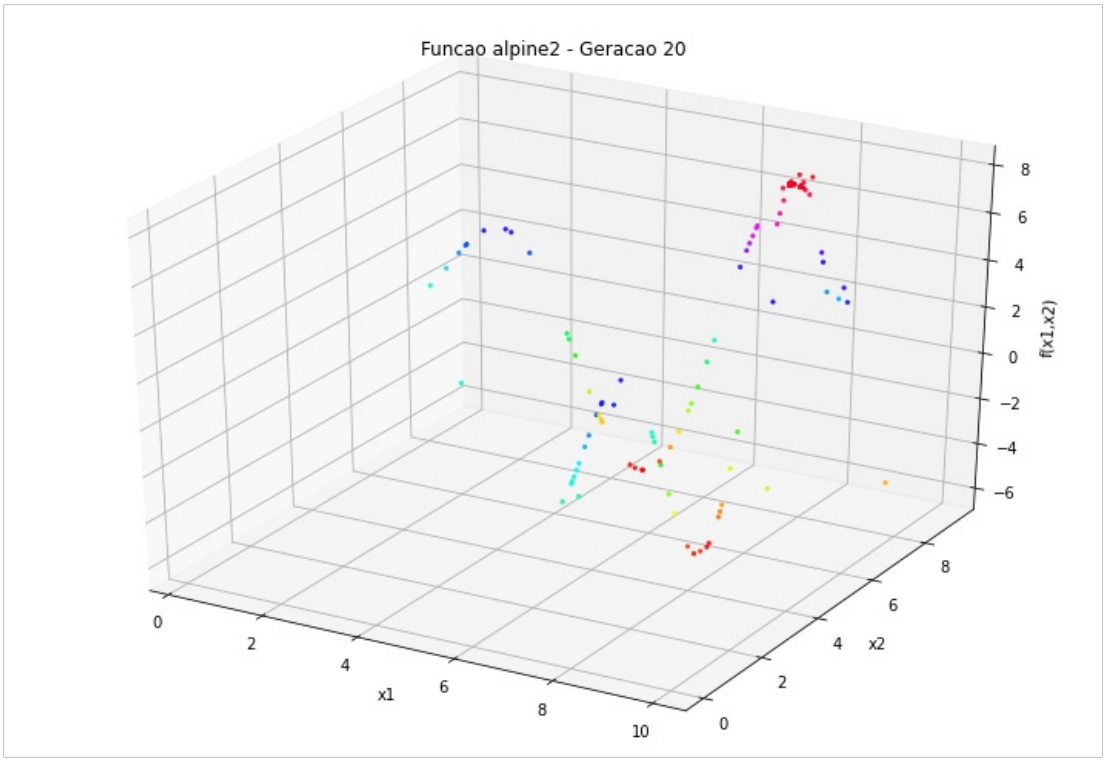

20%|██ | 10/50 [00:54<02:13, 3.35s/it]

Geracao: 10



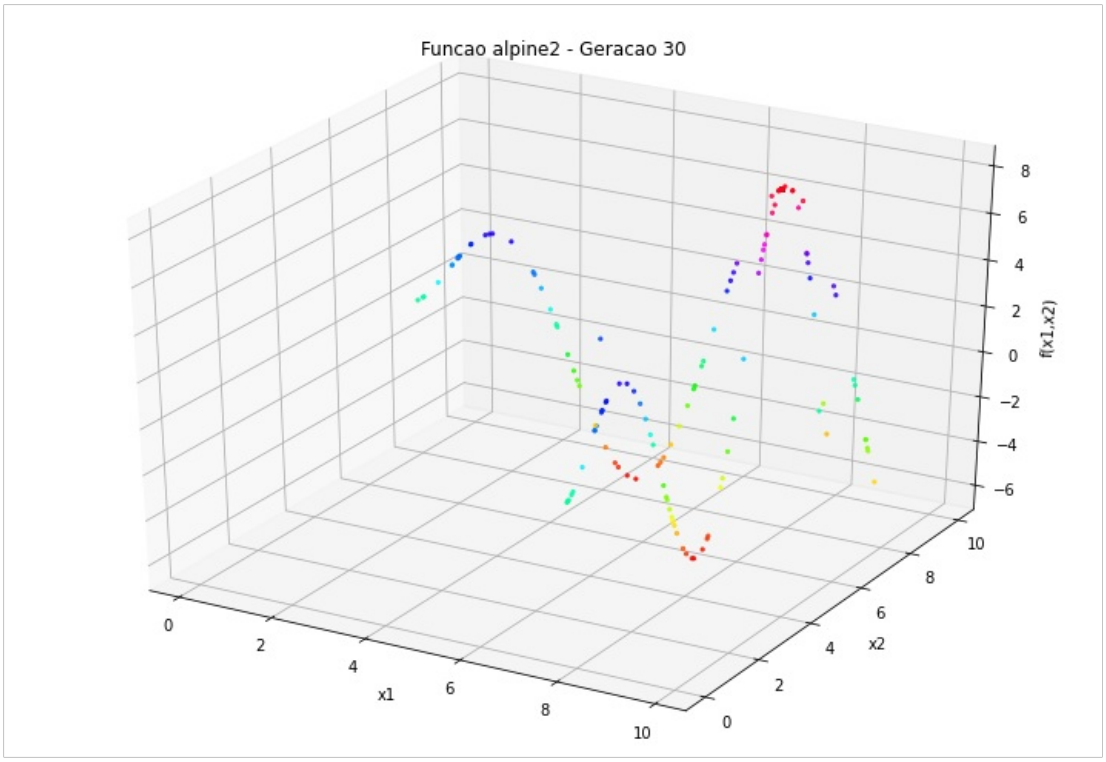

40%|███ | 20/50 [01:15<01:14, 2.50s/it]

Geracao: 20



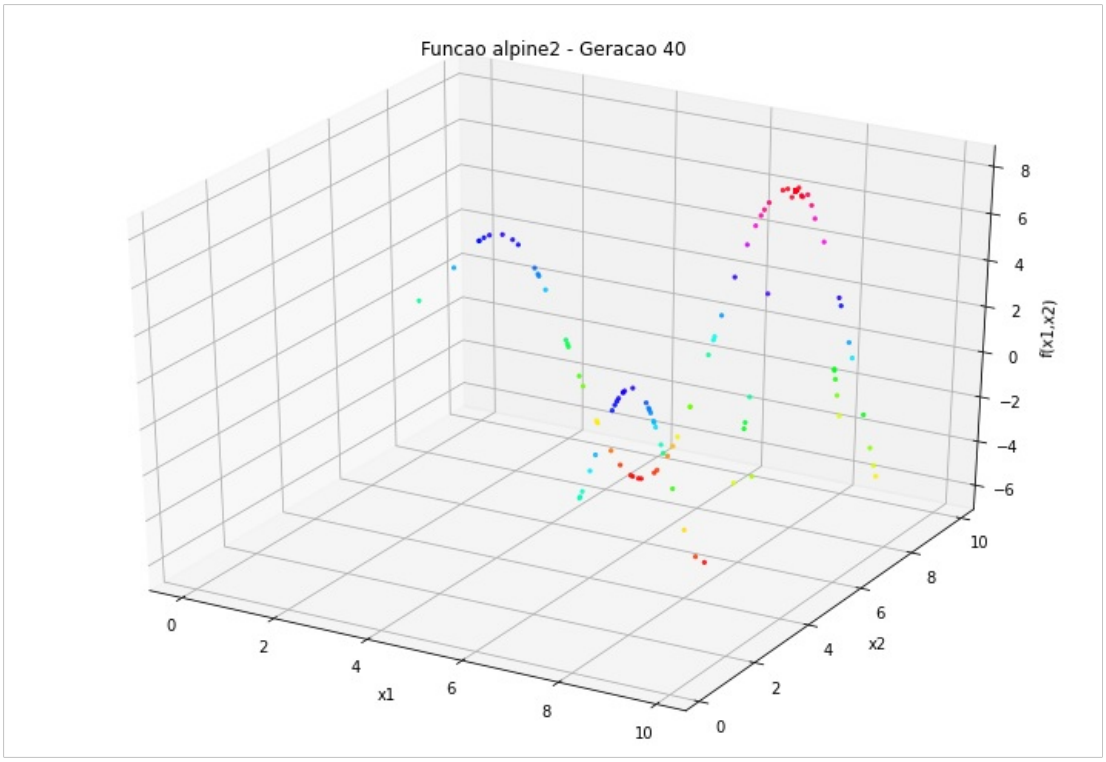

60%|██████    | 30/50 [01:40<00:41,  2.05s/it]

Geracao:  30




80%|████████  | 40/50 [02:01<00:14,  1.47s/it]

Geracao:  40




```
100%|██████████| 50/50 [02:19<00:00,  2.79s/it]

========================================
Best individuo:
X1    7.879557
X2    7.929424
Name: 0, dtype: float64
Funcao alpine02:  7.879389838147665
Geracao:  24
```

# Análises

## Gráfico População por geração

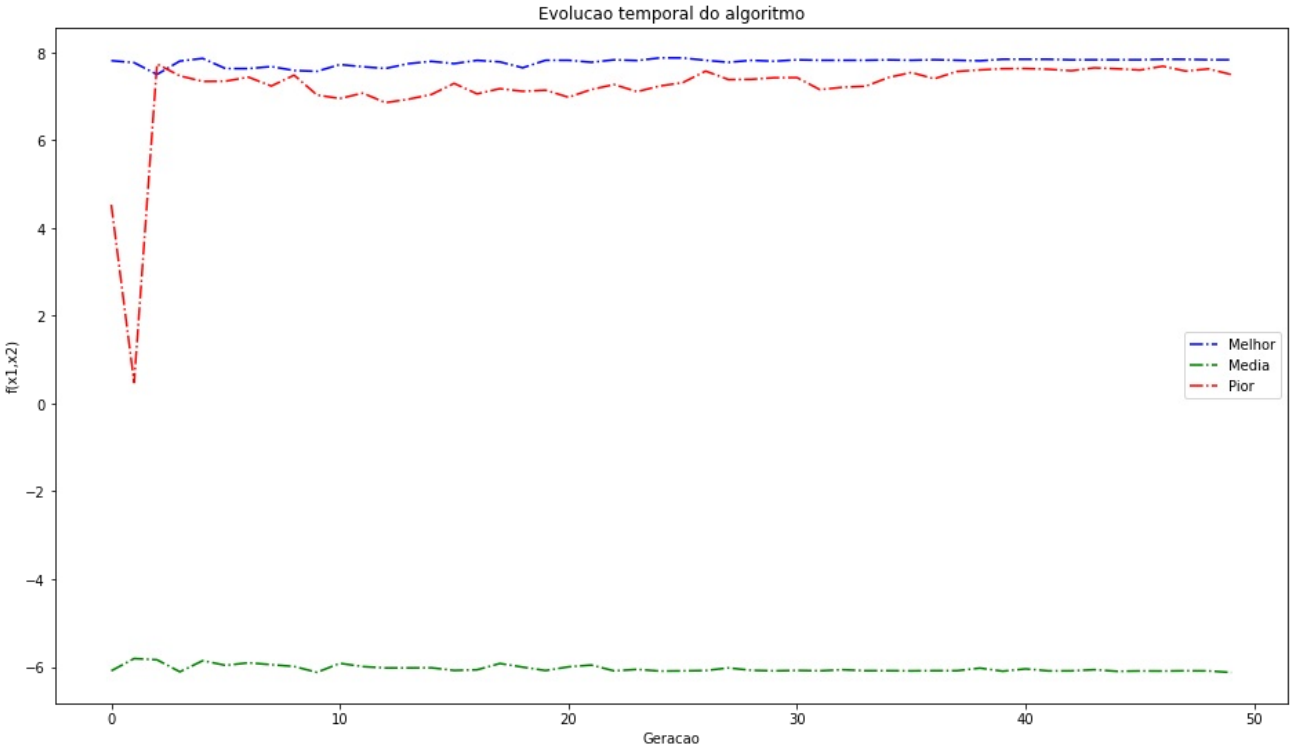
Como podemos ver nos gráficos tridimensionais acima, nosso algorítmo apresentou uma rápida taxa de convergência, atingindo o máximo do espaço de buscas com poucas gerações. Isso aconteceu pois, os melhores indivíduos tem maior prioridades de seleção devido a função triangular aplicada para escolher um valor aleatório. Antes de utilizarmos este método, nosso algorítmo ficava convergino no máximo local em torno de 4.75 nas primeiras gerações. Após a aplicação conseguimos um máximo local proximo a 7.88 na maioria dos casos.

## Gráfico Evolução Temporal do algoritmo

O gráfico abaixo apresenta a evolução temporal do algorítmo genético que implementamos. Neste gráfico podemos notar que houve rápida convergência para o melhor resultado em poucas gerações. É possível ver também que os valores médios e máximos por geração são bem proximos, enquanto os piores estão bem abaixo.

In [50]:

```
plot_evolucao_temporal(melhor,media,pior)
```

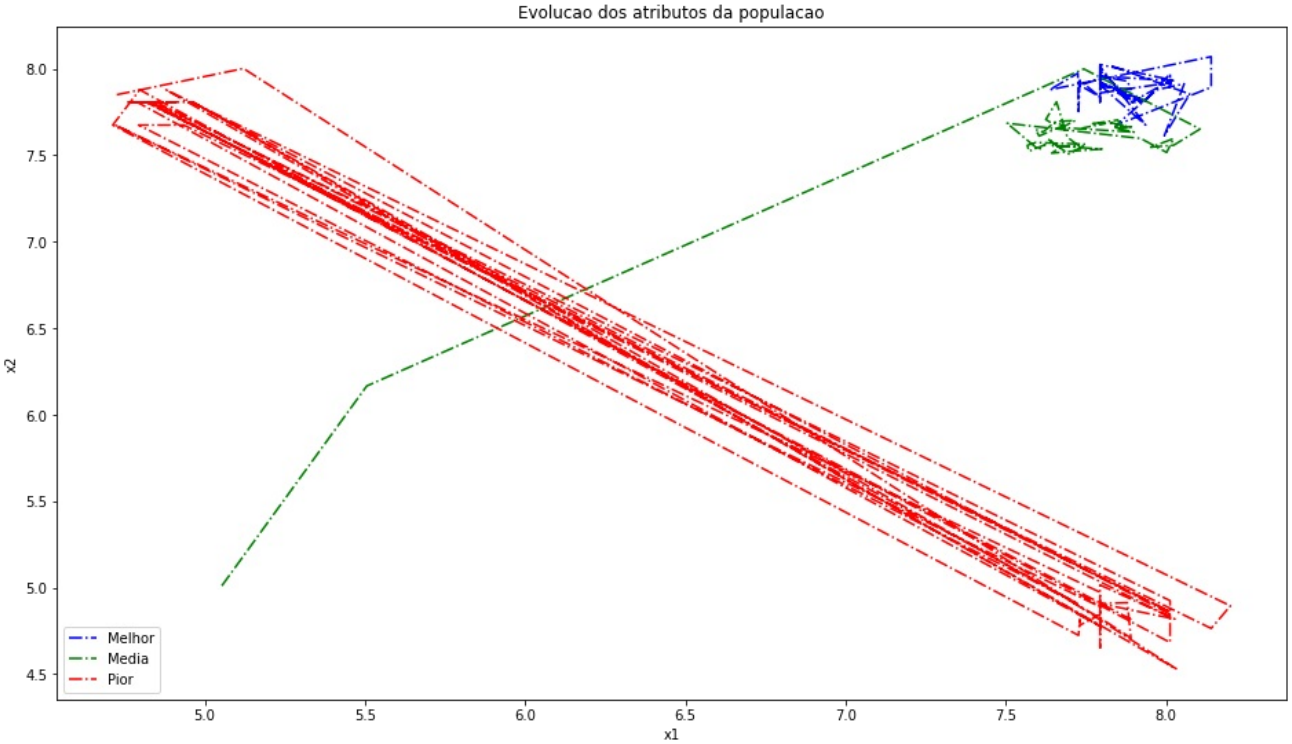



## Gráfico evolução das variáveis por geração

O gráfico abaixo fizemos apenas para testar como as variáveis (x1 e x2) do pior e o melhor indivíduo e o valor médio da população de cada geração ficaram. Cada ponto representa uma geração diferente e os eixos são as variáveis. Algo interessante de notar é que os atributos dos melhores indivíduos apareceram bem próximos, enquanto o pior de cada geração apareceu bastante variado.

In [51]:

```
plot_variaveis_populacao(melhor,pior,media)
```



# Tabela Final

| Genético | |
|---|---|
| Tamanho da população | 1000 |
| Forma de seleção | Roleta |
| Tipo de crossover | Aritmético |
| Função objetivo | Alpine2 |
| Função de Fitness | Ranking Linear |
| Número de Gerações | 50 |
| Taxa de Crossover | 0.8 |
| Taxa de Mutação | 0.05 |