MO431_EX2

April 16, 2021

1 Tarefa 2 - MO431A - 1S2021

Alunos:

- Décio Luiz Gazzoni Filho RA: 264965
- Márcia Jacobina Martins RA: 225269
- Matheus Abrantes Cerqueira RA: 234983

Importando pacotes necessários:

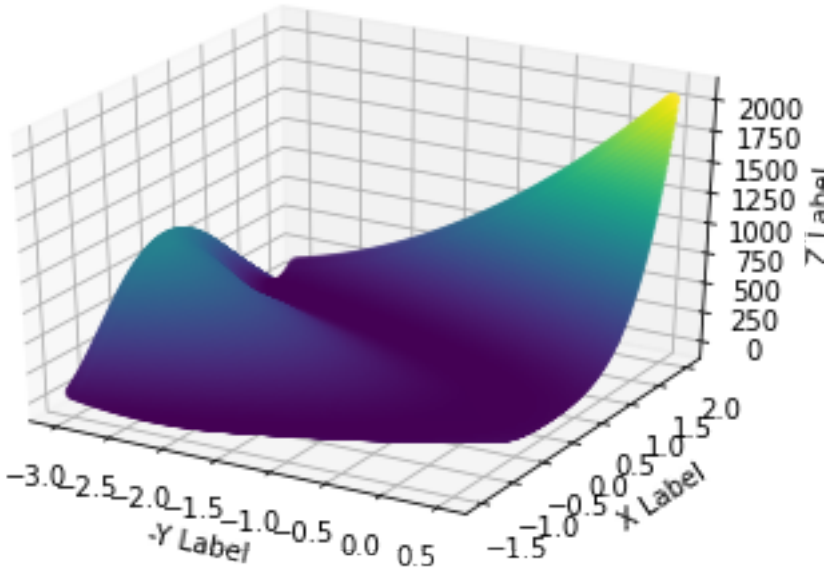
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

Definindo a função de Rosenbrock:

[2]:
```
rosenbrock = lambda x, y: (1 - x)**2 + 100 * (y - x**2)**2
```

Fazendo um gráfico de superfície da função de Rosenbrock:

[3]:
```
passo = 0.02
xmin, xmax = -1.5, 2.0
ymin, ymax = -0.5, 3.0

x, y = np.meshgrid(np.arange(xmin, xmax + passo, passo),
                   np.arange(ymin, ymax + passo, passo))

z = rosenbrock(x,y)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')

ax.scatter(-y, x, z, c=z)

ax.set_xlabel('-Y Label')
ax.set_ylabel('X Label')
ax.set_zlabel('Z Label')

plt.show()
```



2 Questão 1: implementação de descida do gradiente com gradiente explícito

Definindo o gradiente da função de Rosenbrock:

[4]:
```
def grad_rosenbrock(X, Y):
  f1 = lambda x, y: 2 * (200 * x**3 - 200 * x * y + x - 1)
  f2 = lambda x, y: 200 * (y - x**2)

  return f1(X, Y), f2(X, Y)
```

Definindo a função da descida do gradiente, com cálculo explícito do gradiente:

[5]:
```
def grad_descent(lr, tolerancia, max, w0, f, grad, lr_redutor=1.0):
  wi = w0
  wi_logs = []
  f_logs = []
  for i in range(max):
    f_napla_wi = np.array(grad(wi[0], wi[1]))

    wi_1 = wi - lr * f_napla_wi

    custo = np.abs(np.array(f(wi_1[0], wi_1[1])) - np.array(f(wi[0], wi[1])))

    wi_logs.append(wi)
    f_logs.append(np.array(f(wi[0], wi[1])))
    if custo < tolerancia:
      break
```

```
    wi = wi_1

    lr = lr * lr_redutor

  return wi_logs, f_logs
```

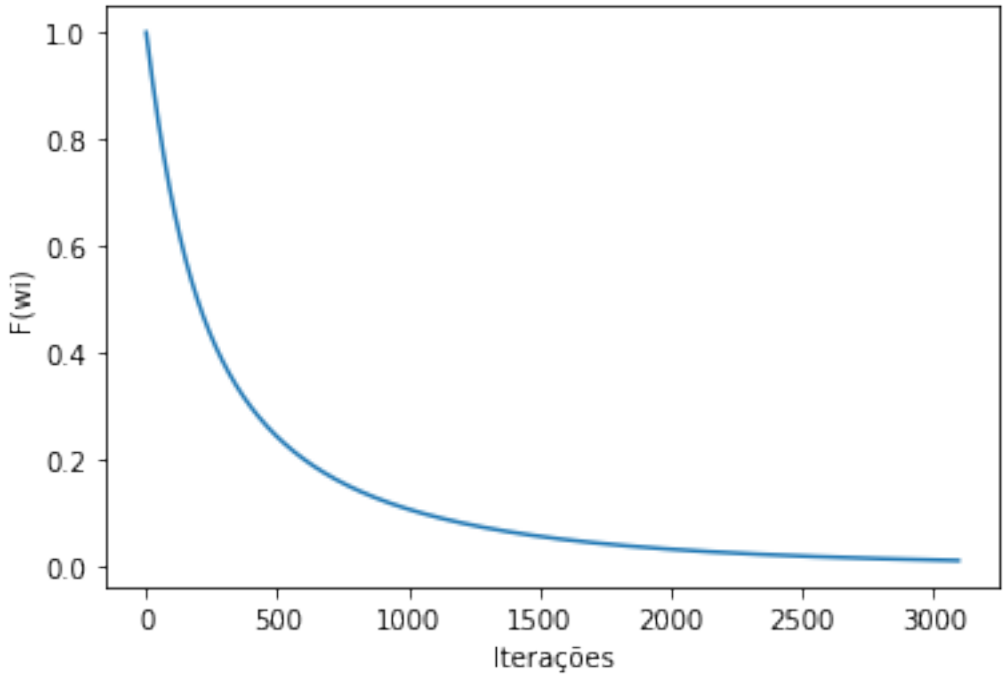
Definindo uma função auxiliar para executar a descida do gradiente com parâmetros fornecidos e traçar um gráfico do resultado:

[6]:
```
def run_grad_descent_and_plot(learning_rate, tol, max_iters=50000,
                              starting_point=np.array([0,0]), func=rosenbrock,
                              grad_func=grad_rosenbrock, redutor=1.0):
  w_lr_logs, f_lr_logs = grad_descent(lr=learning_rate, tolerancia=tol,
                                      max=max_iters, w0=starting_point, f=func,
                                      grad=grad_func, lr_redutor=redutor)

  print("Demorou ", len(w_lr_logs), "iteracoes, último ponto: ", w_lr_logs[-1])

  plt.plot(f_lr_logs)
  plt.ylabel('F(wi)')
  plt.xlabel('Iterações')
  plt.show()

  return w_lr_logs, f_lr_logs
```

2.1 Questão 1.1: use l.r = 1.e-3

[7]:
```
_, f_lr1_logs = run_grad_descent_and_plot(learning_rate=10**-3, tol=10**-5)
```

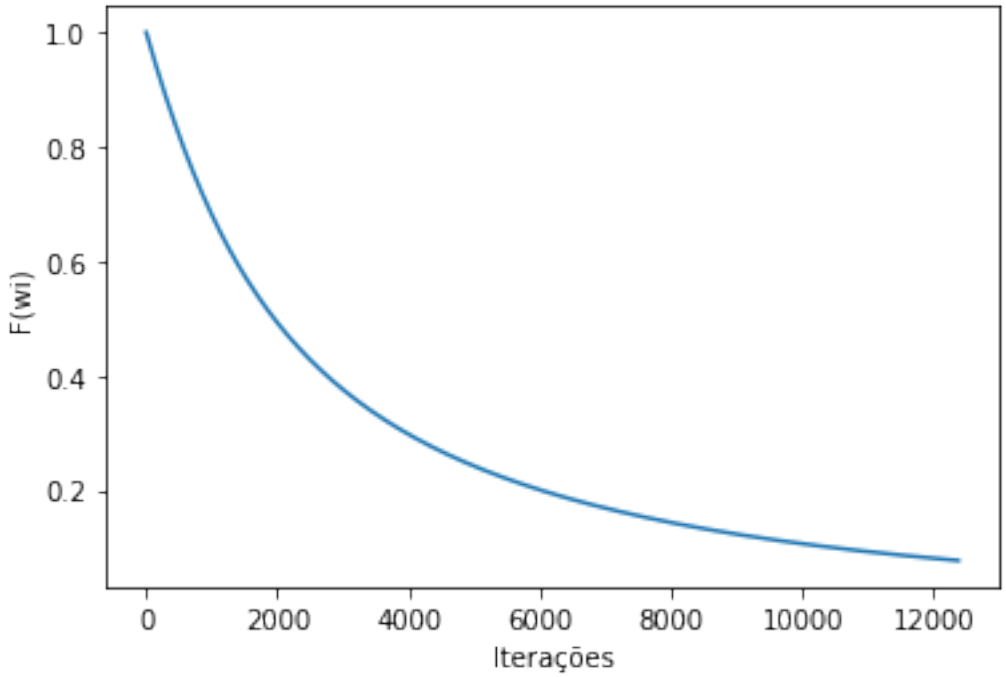
```
Demorou  3096 iteracoes, último ponto:  [0.89731737 0.8047416 ]
```



2.2 Questão 1.2: use l.r = 1.e-4

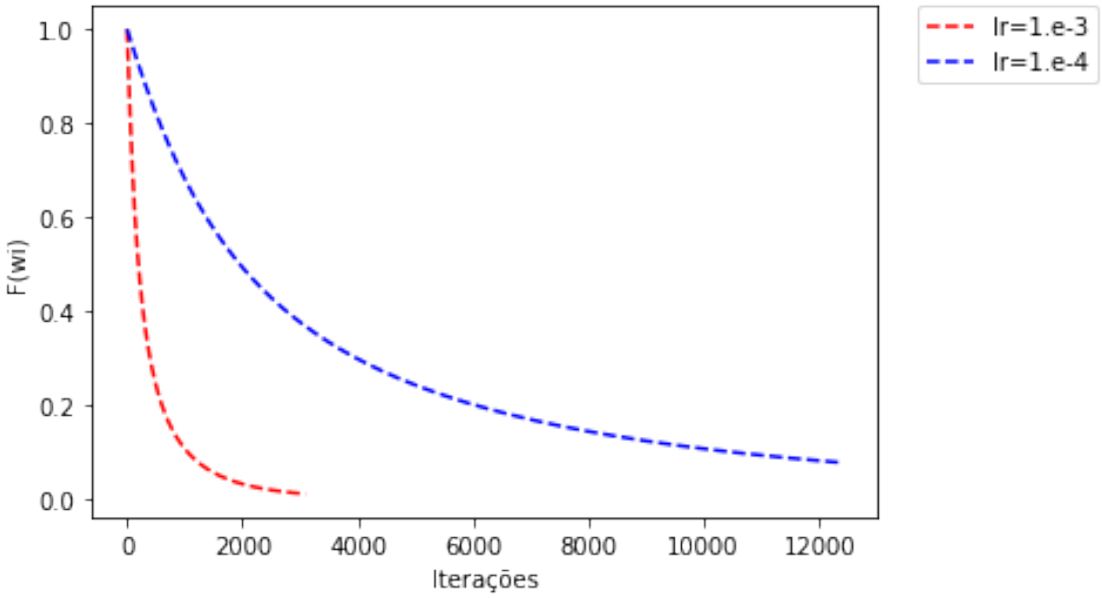
[8]:
```
_, f_lr2_logs = run_grad_descent_and_plot(learning_rate=10**-4, tol=10**-5)
```

```
Demorou  12384 iteracoes, último ponto:  [0.72223396 0.5203204 ]
```



Comparando o desempenho da descida do gradiente com learning rate 1.e-3 e 1.e-4:

[9]:
```
plt.plot(np.linspace(1,len(f_lr1_logs), len(f_lr1_logs), f_lr1_logs,
         'r--', label='lr=1.e-3')
plt.plot(np.linspace(1,len(f_lr2_logs), len(f_lr2_logs), f_lr2_logs,
         'b--', label='lr=1.e-4')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.ylabel('F(wi)')
plt.xlabel('Iterações')
plt.show()
```



Constata-se que o learning rate 1.e-3 convergiu mais rápido.

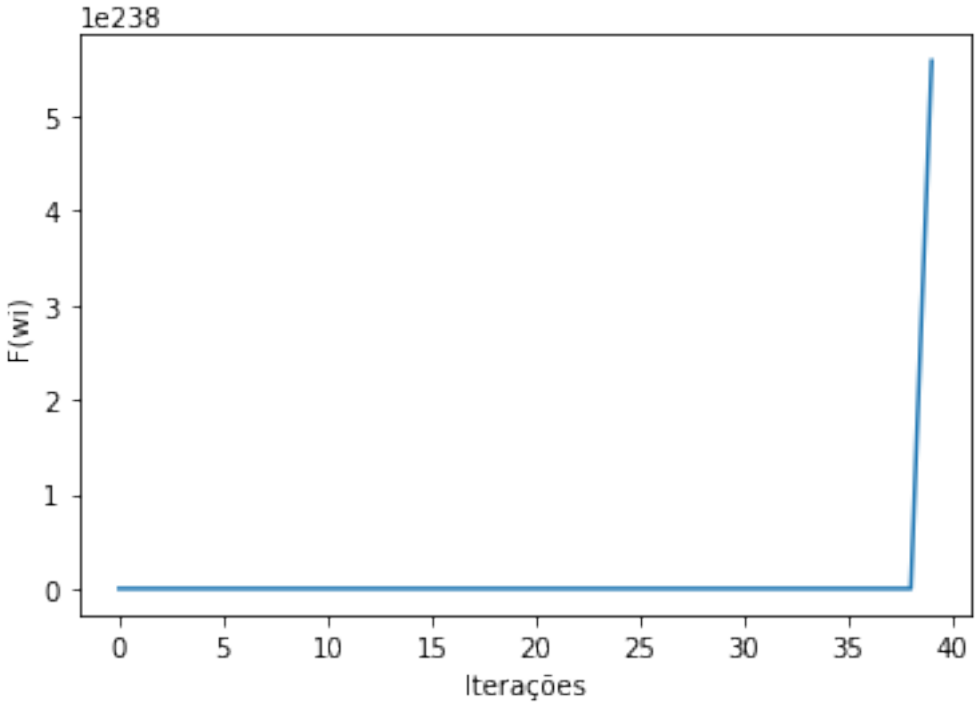
2.3 Questão 1.3: l.r grande

Para um lr grande (lr = 0.01), é possível ver que o problema não converge. Além disso, há diversos casos de overflow no processo:

[10]:
```
_, _ = run_grad_descent_and_plot(learning_rate=10**-2, tol=10**-5)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning:
overflow encountered in double_scalars
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarning:
overflow encountered in double_scalars

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: RuntimeWarning:
overflow encountered in double_scalars
  This is separate from the ipykernel package so we can avoid doing imports
until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning:
invalid value encountered in double_scalars
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarning:
invalid value encountered in double_scalars

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: RuntimeWarning:
invalid value encountered in double_scalars
  This is separate from the ipykernel package so we can avoid doing imports
until
```

```
Demorou  50000 iteracoes, último ponto:  [nan nan]
```

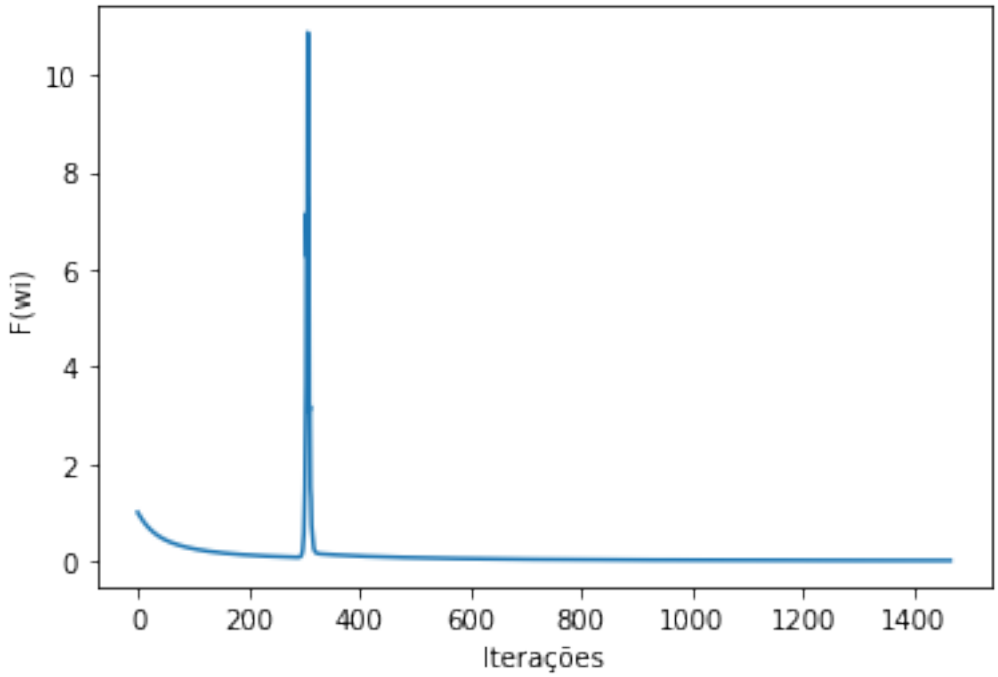


2.4 Questão 1.4: política de redução do l.r

Executando a descida do gradiente com uma política de redução do learning rate (valor inicial 5.e-3 e redutor 0.999), observa-se uma instabilidade em torno da iteração 300 do processo.

[11]:
```
w_lr4_logs, _ = run_grad_descent_and_plot(learning_rate=5*10**-3, redutor=0.
↪999, tol=10**-5)
```

```
Demorou  1468 iteracoes, último ponto:  [0.90380234 0.81645096]
```



Para melhor visualizar esta instabilidade, serão traçados três gráficos:

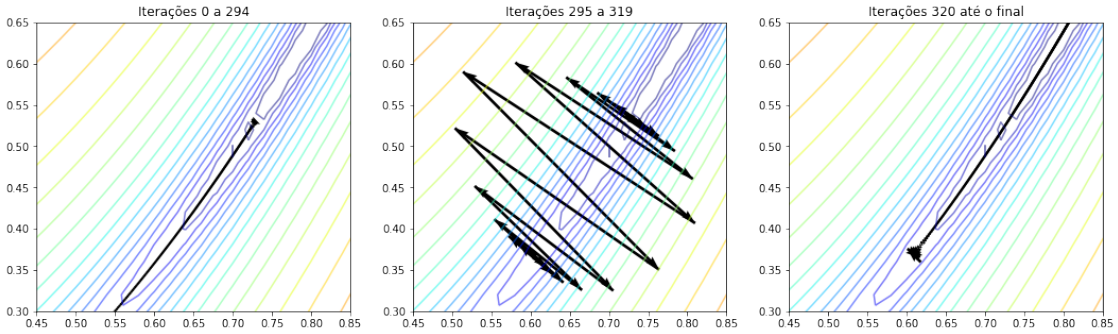
1. da iteração 0 até 294 (gráfico da esquerda);
2. da iteração 295 até 319 (gráfico do meio);
3. da iteração 320 até o final (gráfico da direita).

Observa-se que, um pouco antes da iteração 294, a descida do gradiente apresenta uma certa instabilidade, que se amplifica durante as iterações 295 até 319, retornando para um ponto atingido durante as iterações anteriores.

A partir da iteração 320, a instabilidade se amortece e a descida da gradiente percorre novamente o mesmo caminho anterior. Porém, ao chegar ao ponto em que ocorreu a instabilidade anteriormente, o learning rate é menor, em função da política de redução deste parâmetro. Devido a esta redução, não ocorre mais o fenômeno de instabilidade observado anteriormente, permitindo que a descida do gradiente continue até atingir a tolerância solicitada.

[12]:
```
from matplotlib.colors import LogNorm

fig, axs = plt.subplots(1, 3, figsize=(18, 5))

path = np.array(w_lr4_logs)
indexes = np.linspace(0, len(w_lr4_logs), num=len(w_lr4_logs), endpoint=False)
paths = [path[0:295].T, path[295:320].T, path[320:].T]
titles = ["Iterações 0 a 294",
          "Iterações 295 a 319",
          "Iterações 320 até o final"]

for i,ax in enumerate(axs):
```

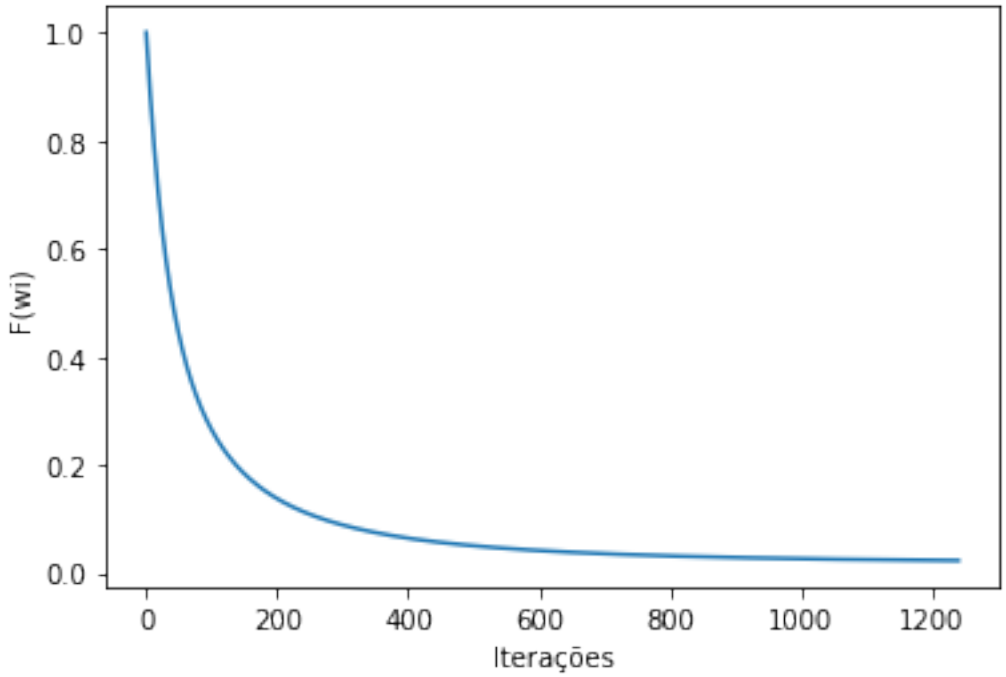
```
  ax.contour(x, y, z, levels=np.logspace(-1, 2, 20), norm=LogNorm(),
                     cmap=plt.cm.jet, alpha=0.5)
  ax.quiver(paths[i][0,:-1], paths[i][1,:-1],
            paths[i][0,1:]-paths[i][0,:-1], paths[i][1,1:]-paths[i][1,:-1],
            scale_units='xy', angles='xy', scale=1.0, width=0.009, color=["k"] )
  ax.set_xlim((0.45, 0.85))
  ax.set_ylim((0.3, 0.65))
  ax.set_title(titles[i])

plt.show()
```



Por curiosidade, foi testada uma política de redução do learning rate mais "agressiva" (taxa de redução de 0.998 ao invés de 0.999). Nesta situação, não ocorre a instabilidade anterior. A hipótese é que o learning rate atingido nas iterações problemáticas (em torno de 300) já estava consideravelmente menor do que no caso em que o redutor é 0.999, evitando a instabilidade. Além disso, houve uma convergência mais rápida.

[13]:
```
_, _ = run_grad_descent_and_plot(learning_rate=5*10**-3, redutor=0.998,␣
↪tol=10**-5)
```

```
Demorou  1241 iteracoes, último ponto:  [0.84771742 0.71795797]
```



3 Questão 2: usando o TensorFlow para calcular o gradiente

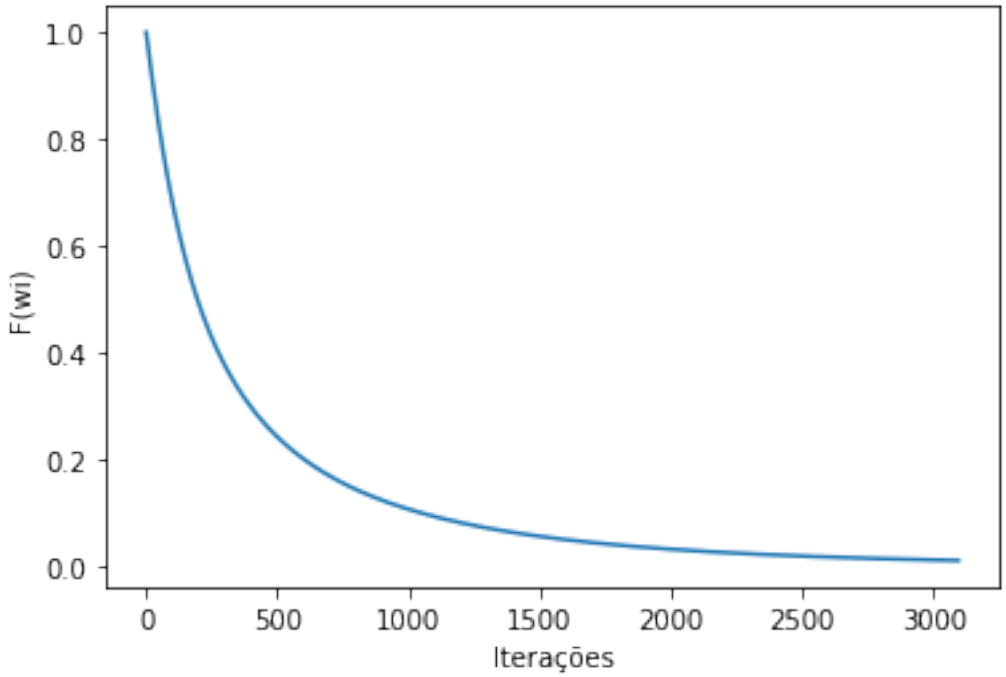
Definindo uma função de gradiente que usa o TensorFlow:

[14]:
```
def grad_rosenbrock_tf(X, Y):
    x = tf.Variable(float(X))
    y = tf.Variable(float(Y))
    with tf.GradientTape() as tape:
      z = rosenbrock(x, y)

    gr = tape.gradient(z, [x, y])

    return gr[0].numpy(), gr[1].numpy()
```

Recalculando a descida do gradiente, passando a função que calcula o gradiente usando TensorFlow, e considerando o learning rate que deu o melhor resultado entre os anteriores (1.e-3):

[15]:
```
_, _ = run_grad_descent_and_plot(learning_rate=10**-3, tol=10**-5,
                                 grad_func=grad_rosenbrock_tf)
```

```
Demorou  3096 iteracoes, último ponto:  [0.89731735 0.80474162]
```



Comparando com a execução anterior (usando o gradiente explícito), observa-se que o mesmo resultado foi obtido, incluindo o mesmo número de iterações. Há apenas uma pequena diferença na última casa decimal de cada coordenada, o que provavelmente pode ser atribuído a erros de arredondamento de diferentes cálculos em ponto flutuante realizados em cada caso.