MO431A_Tarefa2

April 15, 2021

MO431A - Fundamentos de Álgebra Linear e Otimização para Aprendizado de Máquina
Equipe:

- Maria Fernanda Tejada Begazo - RA 197488
- Jose Italo da Costa Silva - RA 265682
- Gian Franco Joel Condori Luna - RA 234826

Tarefa 02

A tarefa foi desenvolvida na linguagem python. Para isso utilizou-se notebooks jupyter no ambiente Google Colaboratory (Google Colab).

Codificação:

[105]:
```
#Primeiro faz-se os imports necessários:
import numpy as np
from numpy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.colors import LogNorm
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sympy import Derivative, diff, simplify
from sympy import Symbol, exp, Heaviside
import scipy.linalg as sciLa
from autograd import elementwise_grad, value_and_grad

import pandas as pd
import cv2
import io
import tensorflow as tf
```

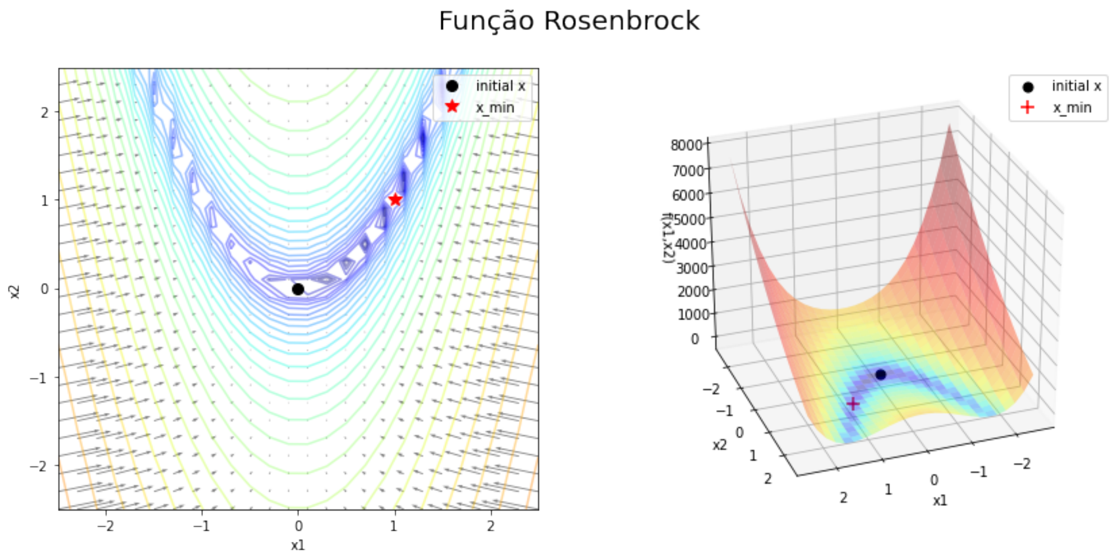
Nós vamos a minimizar a funcção de 2 dimensões:
$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$

[106]:
```
def f(X1, X2):
  return (1-X1)**2 + 100*(X2 - X1**2)**2

def error(x_velho, x_novo):
  return np.abs(f(x_novo[0], x_novo[1]) - f(x_velho[0], x_velho[1]))
```

[282]:
```
# Data for a three-dimensional line
def functionPlot(title='', data = None, xmin = None):
  x_min = 2.5
  x = np.arange(-x_min, x_min+0.2, 0.2)
  y = np.arange(-x_min, x_min+0.2, 0.2)
  X, Y = np.meshgrid(x, y)
  Z = f(X, Y)

  fig = plt.figure(figsize=(14,6))
  cmp = plt.cm.jet

  # Curva de Nivel
  ax = fig.add_subplot(1, 2, 1)
  dz_dx = elementwise_grad(f, argnum=0)(X, Y)
  dz_dy = elementwise_grad(f, argnum=1)(X, Y)
  ax.contour(X, Y, Z, levels=np.logspace(0, 5, 35), norm=LogNorm(), cmap=cmp,␣
↪alpha=.4)
  ax.quiver(X, Y, X - dz_dx, Y - dz_dy, alpha=.5)
  ax.plot(0,0, 'ko', markersize=8, label='initial x')
  ax.plot(1,1, 'r*', markersize=10, label='x_min')
  if data is not None:
    for i in range(len(data)):
      ax.plot(data[i,0], data[i,1], 'k', marker=7, markersize=8 )

  if (xmin is not None):
    ax.plot(xmin[0], xmin[1],'m*', markersize=10, label='Find x_min')

  ax.set_xlabel('x1')
  ax.set_ylabel('x2')
  ax.legend()

  # Imagem em 3D
  ax = fig.add_subplot(1, 2, 2, projection='3d')

  ax.plot_surface(X,Y, Z, norm=LogNorm(), rstride=1, cstride=1,␣
↪edgecolor='none', alpha=.4, cmap=cmp)

  # Data for three-dimensional scattered points
  ax.scatter(0, 0, f(0,0), color='k', marker='o', s=50, label='initial x')
  ax.scatter(1, 1, f(1,1), color='r', marker='+', s=100, label='x_min')

  if data is not None:
    for i in range(len(data)):
      ax.scatter3D(data[i,0], data[i,1], f(data[i,0],data[i,1]), color='k',␣
↪marker=7, s=80)

  if (xmin is not None):
```

```
    ax.scatter3D(xmin[0], xmin[1], f(xmin[0],xmin[1]), color='m', marker='*',␣
↪s=80, label='Find x_min')

  ax.set_xlabel('x1')
  ax.set_ylabel('x2')
  ax.set_zlabel('f(x1,x2)')
  ax.legend()
  ax.view_init(30, 70)

  plt.suptitle(title, size=20)
  plt.show()
```

[283]:
```
functionPlot(title = "Função Rosenbrock")
```



1 Implementação de descida do gradiente com gradiente explicito

[260]:
```
#Função que computa o gradiente de f
x1 = Symbol('x1')
x2 = Symbol('x2')
fx = (1-x1)**2 + 100*(x2 - x1**2)**2
dx1 = Derivative(fx, x1).doit()
dx2 = Derivative(fx, x2).doit()
print("Derivative of x1: ", dx1)
print("Derivative of x2: ", dx2)
```

```
Derivative of x1:  -400*x1*(-x1**2 + x2) + 2*x1 - 2
Derivative of x2:  -200*x1**2 + 200*x2
```

[261]:
```
def gradiente(x):
  dx1 = -400*x[0]*(-x[0]**2 + x[1]) + 2*x[0] - 2
  dx2 = -200*x[0]**2 + 200*x[1]
  return np.array([dx1, dx2])
```

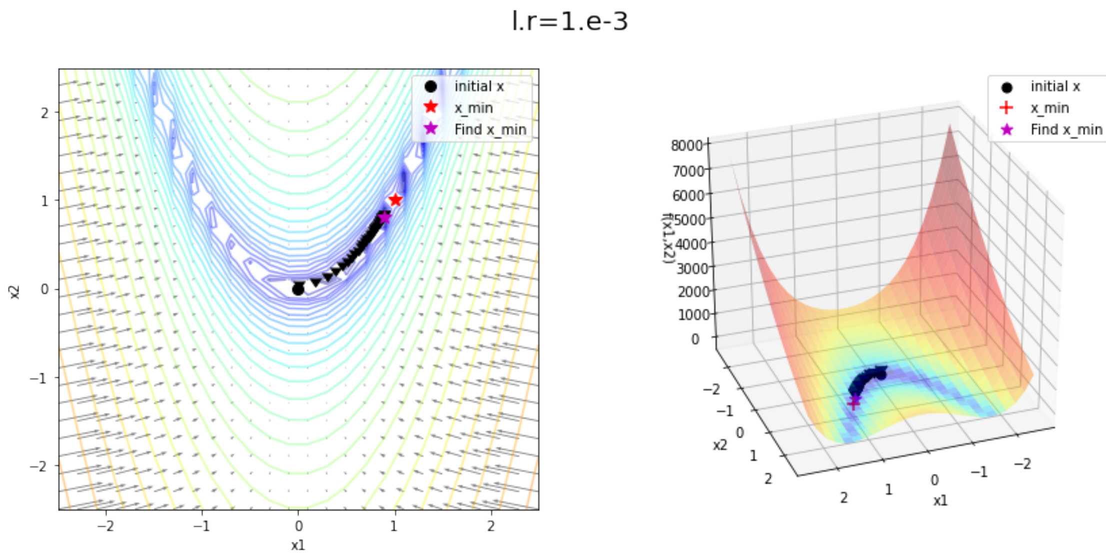
[262]:
```
def descidaGradiente(x_, learn_rate, tolerancia, max_steps):
  conver = 9999.0
  steps = 0
  x = np.copy(x_)
  xGrad = []
  while (conver > tolerancia) and (steps < max_steps):
    xnew = x - learn_rate * gradiente(x)
    conver = error(x, xnew)
    x = xnew
    xGrad.append(x)
    steps = steps + 1

  print("x_min = ", x)
  print("f_min = ", np.round(f(x[0], x[1]),3) )
  print("Nro Steps = ", steps)
  return x, np.array(xGrad)
```

1.1 Use l.r = 1.e-3

[290]:
```
x = np.array([0,0])
nroMaxPassos = 50000
tolerancia = 0.00001
learn_rate = 0.001
x_min, xGrad = descidaGradiente(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  [0.89736593 0.80482897]
f_min =  0.011
Nro Steps =  3096
```

[291]:
```
functionPlot('l.r=1.e-3', xGrad[0:len(xGrad):100], x_min)
```

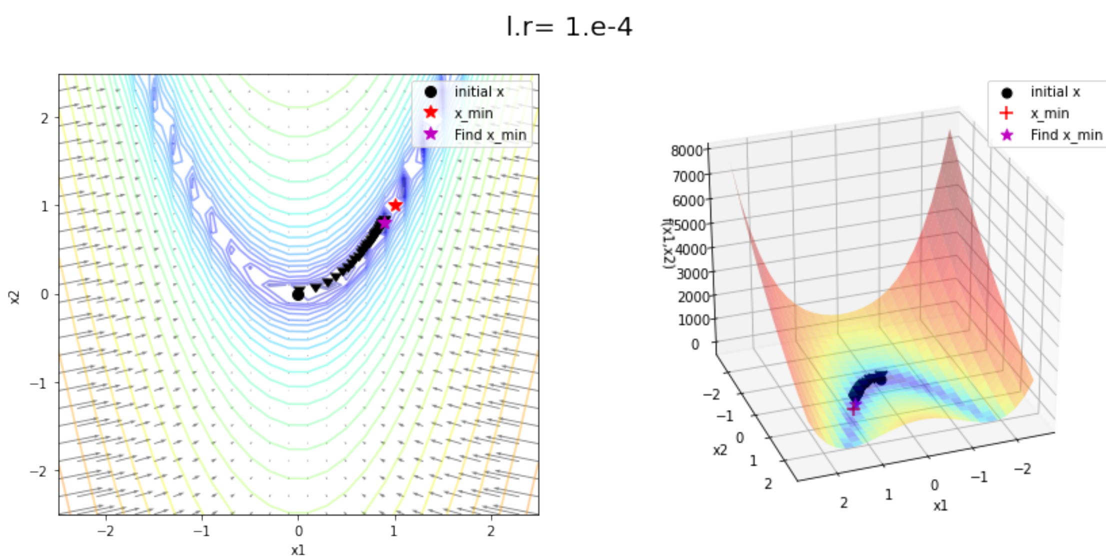


1.2 Use l.r = 1.e-4

[311]:
```
learn_rate = 0.0001
x_min, xGrad = descidaGradiente(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  [0.72225191 0.52034643]
f_min =  0.077
Nro Steps =  12384
```

[292]:
```
functionPlot('l.r= 1.e-4', xGrad[0:len(xGrad):100], x_min)
```



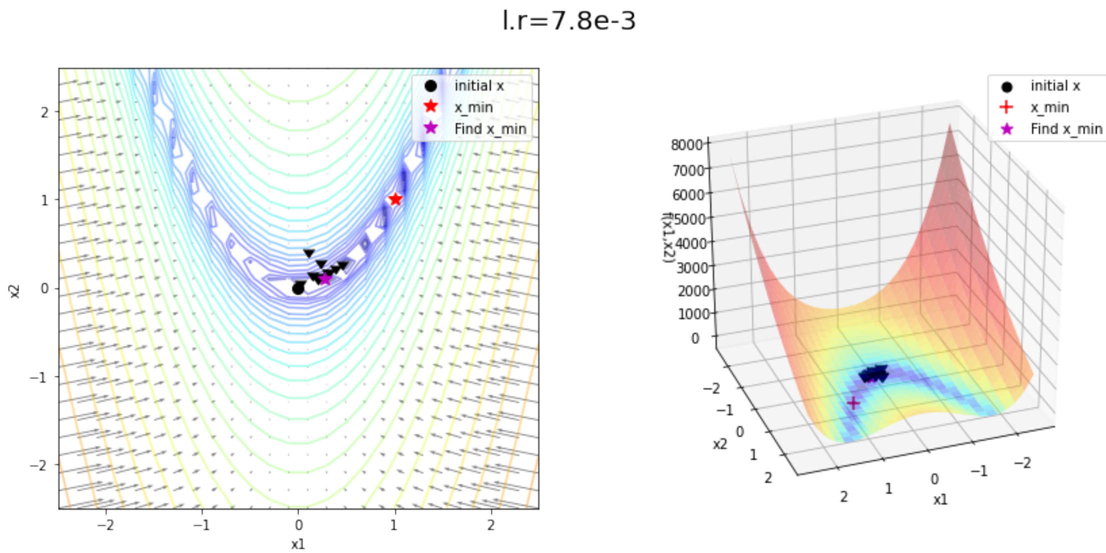
1.3 Use l.r grande

1.3.1 l.r = 0.0078

[310]:
```
learn_rate = 0.0078
x_min, xGrad = descidaGradiente(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  [0.27380053 0.10092881]
f_min =  0.595
Nro Steps =  499
```

[294]:
```
functionPlot('l.r=7.8e-3', xGrad[0:len(xGrad):50], x_min)
```



1.3.2 l.r. = 1.e-2

[309]:
```
learn_rate = 0.01
x_min, xGrad = descidaGradiente(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  [-inf  inf]
f_min =  nan
Nro Steps =  41

/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
```

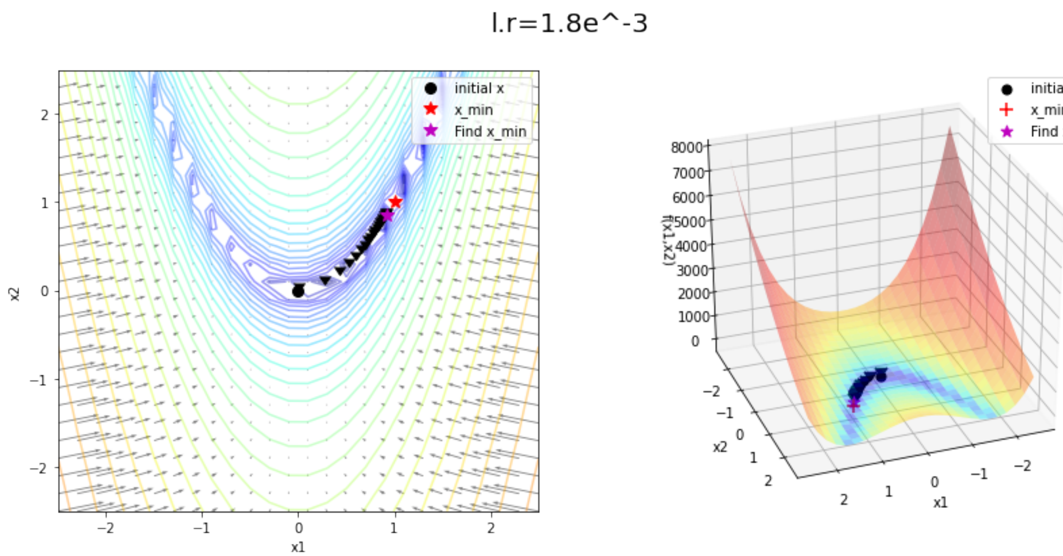
```
until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarning:
invalid value encountered in double_scalars
```

1.3.3 l.r = 1.8e-3

[308]:
```
learn_rate = 0.0018
x_min, xGrad = descidaGradiente(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  [0.92188955 0.84955281]
f_min =  0.006
Nro Steps =  2047
```

[298]:
```
functionPlot('l.r=1.8e^-3', xGrad[0:len(xGrad):100], x_min)
```



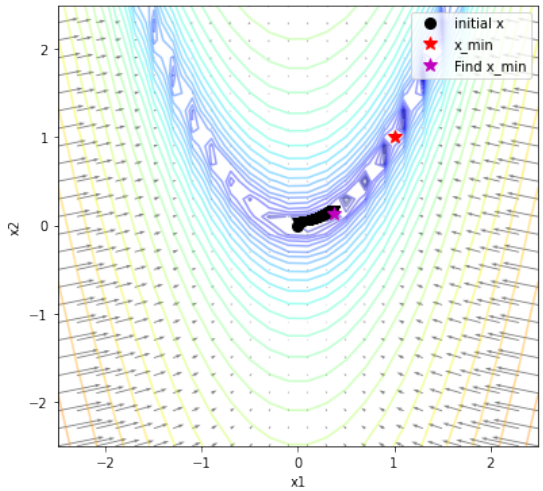
1.3.4 l.r = 1.e-5

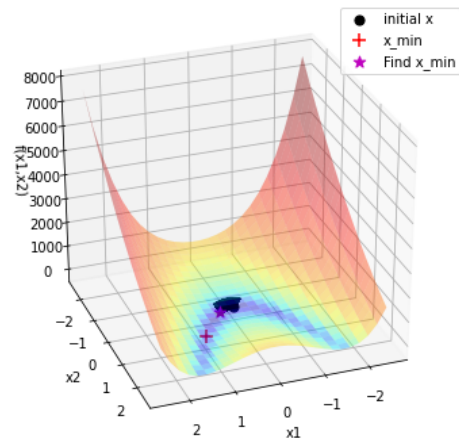
[307]:
```
learn_rate = 0.00001
x_min, xGrad = descidaGradiente(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  [0.37501428 0.1376293 ]
f_min =  0.392
Nro Steps =  28405
```

[302]:
```
functionPlot('l.r.=1.e-5', xGrad[0:len(xGrad):100], x_min)
```

l.r.=1.e-5





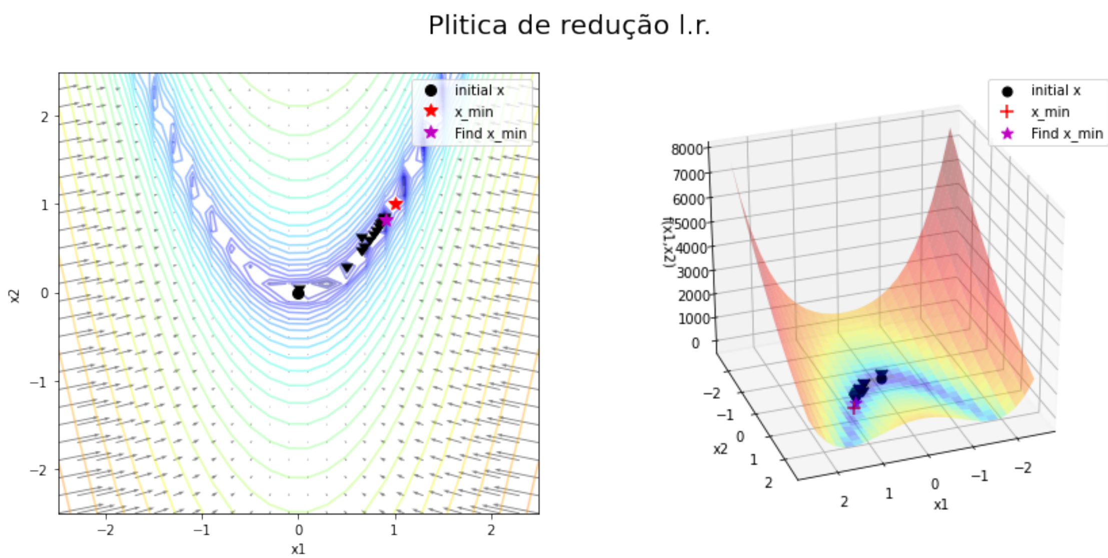
1.4 Politica de redução do l.r

[303]:
```
def descidaGradienteMod(x_, learn_rate, beta, tolerancia, max_steps):
  conver = 9999.0
  steps = 0
  x = np.copy(x_)
  xGrad = []
  while (conver > tolerancia) and (steps < max_steps):
    xnew = x - learn_rate * gradiente(x)
    conver = error(x, xnew)
    x = xnew
    learn_rate = beta*learn_rate
    xGrad.append(x)
    steps = steps + 1

  print("x_min = ", x)
  print("f_min = ", np.round(f(x[0], x[1]),3) )
  print("Nro Steps = ", steps)
  return x, np.array(xGrad)
```

[304]:
```
x = np.array([0,0])
nroMaxPassos = 50000
tolerancia = 0.00001
learn_rate = 0.005
beta = 0.999
x_min, xGrad = descidaGradienteMod(x, learn_rate, beta, tolerancia,␣
 ↪nroMaxPassos)
```

```
x_min =  [0.90394626 0.81671177]
f_min =  0.009
Nro Steps =  1467
```

[306]:
```
functionPlot('Plitica de redução l.r.', xGrad[0:len(xGrad):100], x_min)
```



2 Usando do Tensorflow para calcular o gradiente

[312]:
```
(1-x1)**2 + 100*(x2 - x1**2)**2
def func(x1, x2):
  return (1 - x1)**2 + 100 * (x2 - x1**2)**2
```

[313]:
```
x1 = tf.Variable(0.0, trainable=True, dtype=tf.float64, name='x1')
x2 = tf.Variable(0.0, trainable=True, dtype=tf.float64, name='x2')

def objective():
  return (1 - x1)**2 + 100 * (x2 - x1**2)**2
```

[314]:
```
def optimize(start, lrate, tolerancia, max_steps, beta=1.0):
  x1.assign(start[0])
  x2.assign(start[1])
  conver = 99999.0
  steps = 0

  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
      initial_learning_rate=lrate,
      decay_steps = 1,
      decay_rate= beta
```

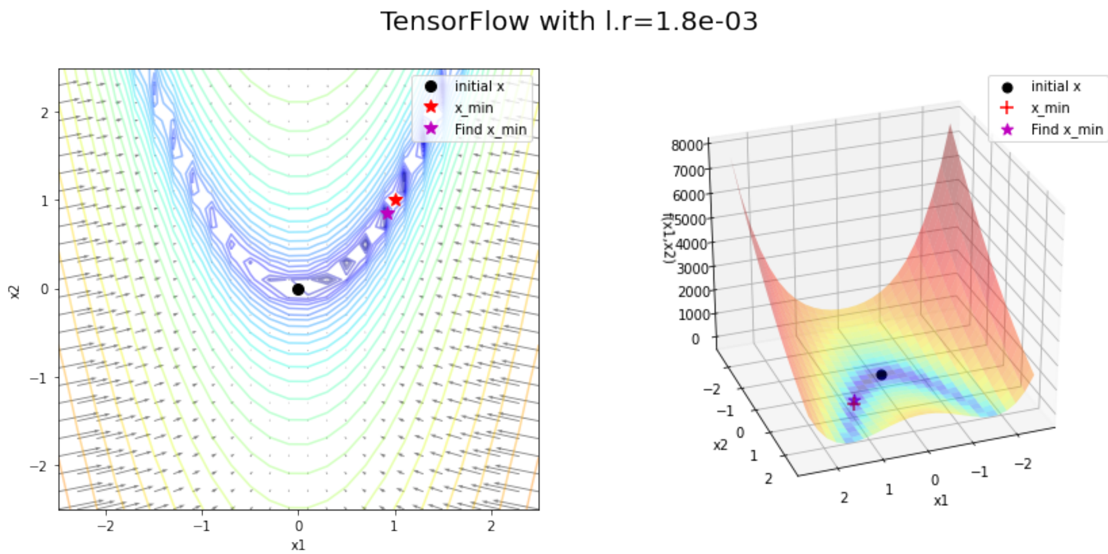
```
)
opt = tf.keras.optimizers.SGD(learning_rate=lr_schedule)

obj_vals = []
coords = [[9999, 9999]]

while (conver > tolerancia) and (steps < max_steps):
  obj_vals.append(objective().numpy())
  coords.append((x1.numpy(), x2.numpy()))
  opt.minimize(objective, var_list=[x1, x2])
  steps = steps+1
  conver = error(coords[steps-1], coords[steps])

coords[0] = (0.0,0.0)
print("x_min = ", coords[steps])
print("f_min = ", np.round(f(coords[steps][0], coords[steps][1]),3) )
print("Nro Steps = ", steps)

return coords[steps], coords
```

2.1 Use l.r =1.8e-03

[315]:
```
x = np.array([0,0])
nroMaxPassos = 50000
tolerancia = 0.00001
learn_rate = 0.0018
x_min, coords = optimize(x, learn_rate, tolerancia, nroMaxPassos)
```

```
x_min =  (0.9218895488032104, 0.8495528107001615)
f_min =  0.006
Nro Steps =  2048
```

[317]:
```
functionPlot("TensorFlow with l.r=1.8e-03", None, x_min)
```



2.2 Use ExponentialDecay

[318]:
```
x = np.array([0,0])
nroMaxPassos = 50000
tolerancia = 0.00001
learn_rate = 0.0018
beta = 0.9999
x_min, coords = optimize(x, learn_rate, tolerancia, nroMaxPassos, beta)
```

```
x_min =  (0.9136895194072447, 0.8344648605082549)
f_min =  0.007
Nro Steps =  2140
```

[319]:
```
functionPlot('TensorFlow with Exponential Decay of l.r', None, x_min)
```

TensorFlow with Exponential Decay of l.r