In [51]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar, fmin_l_bfgs_b, minimize, line_search
!pip install py-bobyqa
import pybobyqa
```

```
Requirement already satisfied: py-bobyqa in /usr/local/lib/python3.
6/dist-packages (1.1.1)
Requirement already satisfied: pandas>=0.17 in /usr/local/lib/python
3.6/dist-packages (from py-bobyqa) (0.23.4)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python
3.6/dist-packages (from py-bobyqa) (1.2.1)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python
3.6/dist-packages (from py-bobyqa) (1.16.2)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python
3.6/dist-packages (from pandas>=0.17->py-bobyqa) (2018.9)
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/
lib/python3.6/dist-packages (from pandas>=0.17->py-bobyqa) (2.5.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/
dist-packages (from python-dateutil>=2.5.0->pandas>=0.17->py-bobyqa)
(1.11.0)
```

Função

$f(x1, x2, x3) = - (100(x1 - x2^2) - (x1 - 1)^2 + 90(x2 - x3^2) - (x2 - 1)^2)$

deriv_x1 = -102+2*x1

deriv_x2 = -92+202*x2

deriv_x3 = +180*x3

In [0]:

```
tol = 0.00001 #precisão ou tolerancia
ln = 0.000001 # Learning rate
initial_values = np.array([0.5, 0.5, 0.5])

count_function_call = 0

def function(x1, x2, x3):
    global count_function_call
    count_function_call += 1
    return -(100*(x1 - (x2**2)) - (x1-1)**2 + 90*(x2-(x3**2)) - (x2 - 1)**2)

def function_param(params):
    x1, x2, x3 = params
    return function(x1, x2, x3)

def deriv_x1(x1):
    return -102+2*x1

def deriv_x2(x2):
    return -92+202*x2

def deriv_x3(x3):
    return 180*x3
```

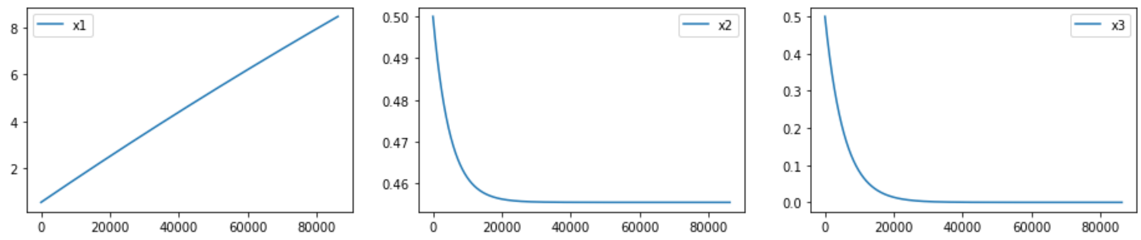
2.1 Descida do gradiente

In [53]:

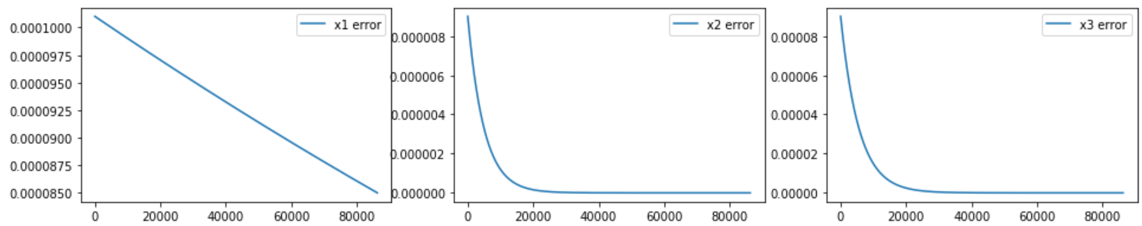
```
def gradient_descent(x1, x2 ,x3, ln, precision):

    xs_hist = []
    precision_hist = []
    y_list = []

    x1_prev = 99999

    while (abs(x1 - x1_prev)/x1_prev) > precision:

        x1_prev = x1
        x2_prev = x2
        x3_prev = x3

        x1 = x1 - (ln * (deriv_x1(x1)))
        x2 = x2 - (ln * (deriv_x2(x2)))
        x3 = x3 - (ln * (deriv_x3(x3)))

        xs_hist.append([x1 ,x2, x3]) #Salva os x's encontrados nessa iteração

        #Histórico do erro
        precision_hist.append([abs(x1 - x1_prev), abs(x2 - x2_prev), abs(x3 - x3
_prev)])

        y_list.append(function(x1, x2, x3))

    xs_hist = np.array(xs_hist)
    precision_hist = np.array(precision_hist)
    print("Valores para os x`s: ", [x1 ,x2, x3])

    iterations = len(xs_hist)

    fig=plt.figure(figsize=(16, 3))
    fig.add_subplot(1, 3, 1)
    plt.plot(range(iterations), xs_hist[:,0], label="x1")
    plt.legend()
    fig.add_subplot(1, 3, 2)
    plt.plot(range(iterations), xs_hist[:,1], label="x2")
    plt.legend()
    fig.add_subplot(1, 3, 3)
    plt.plot(range(iterations), xs_hist[:,2], label="x3")
    plt.legend()
    plt.figure()

    #plot erro relativo
    fig=plt.figure(figsize=(16, 3))
    fig.add_subplot(1, 3, 1)
    plt.plot(range(iterations), precision_hist[:,0], label="x1 error")
    plt.legend()
    fig.add_subplot(1, 3, 2)
    plt.plot(range(iterations), precision_hist[:,1], label="x2 error")
    plt.legend()
    fig.add_subplot(1, 3, 3)
    plt.plot(range(iterations), precision_hist[:,2], label="x3 error")
    plt.legend()
    plt.figure()

    #plot Y
    fig=plt.figure(figsize=(16, 3))
```

```
    plt.plot(range(iterations), y_list[:], label="Y")
    plt.legend()


gradient_descent(0.5, 0.5, 0.5, ln, tol)
```

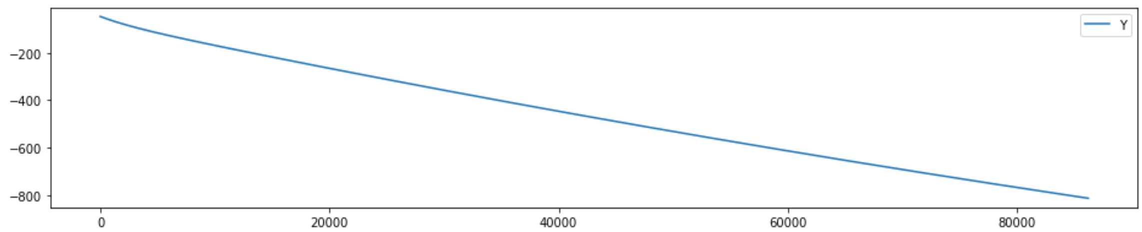
```
Valores para os x`s:  [8.500123764974568, 0.4554455457646628, 9.0580
72712417438e-08]
```




```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```



2.2 Descida do gradiente com busca em linha - Antes da Aula

Aparentemente não faz sentido

In [54]:

```
min_x1 = minimize_scalar(deriv_x1, bounds=(-10, 10), method='Bounded')
min_x2 = minimize_scalar(deriv_x2, bounds=(-10, 10), method='Bounded')
min_x3 = minimize_scalar(deriv_x3, bounds=(-10, 10), method='Bounded')

for params in [(1, str(min_x1.fun), min_x1.nfev),
               (2, str(min_x2.fun), min_x2.nfev),
               (3, str(min_x3.fun), min_x3.nfev)]:
    print("Mínimo X%s : %s, com %s iterações"% params)
```

```
Mínimo X1 : -121.9999867125041, com 31 iterações
Mínimo X2 : -2111.998657962914, com 31 iterações
Mínimo X3 : -1799.998804125369, com 31 iterações
```

2.2 Descida do gradiente com busca em linha - Depois da Aula

In [55]:

```
# função que retorna as tres derivadas
def test_grad(x):
    return [deriv_x1(x[0]), deriv_x2(x[1]), deriv_x3(x[2])]

res = line_search(function_param, test_grad, initial_values, np.array([1, 1, 1
]))
print("Alpha: ", res[0])
print("# chamadas da função: ", res[1])
print("# chamadas da func gradiente: ", res[2])
print(res[3])
print(res[4])
print(res[5])
```

```
Alpha:  0.005208333333333333
# chamadas da função:  8
# chamadas da func gradiente:  1
-47.005208333333336
-47.0
[-100.98958333333333, 10.052083333333343, 90.9375]
```

2.3 L-BFGS

In [56]:

```
min_l_bfgs = minimize(function_param, initial_values, method="L-BFGS-B",
                      bounds=[(-10, 10), (-10, 10), (-10, 10)], tol=tol)

print(min_l_bfgs)
print("Mínimo de F() %s acontece em %s, com %s chamadas a função" %
      (min_l_bfgs.fun, min_l_bfgs.x, min_l_bfgs.nfev))
```

```
      fun: -938.9504950491494
 hess_inv: <3x3 LbfgsInvHessProduct with dtype=float64>
      jac: array([-8.20000082e+01, -3.75166564e-04, -6.82121026e-0
5])
  message: b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
     nfev: 36
      nit: 7
   status: 0
  success: True
        x: array([ 1.00000000e+01,  4.55443700e-01, -3.62461516e-0
7])
Mínimo de F() -938.9504950491494 acontece em [ 1.00000000e+01  4.554
43700e-01 -3.62461516e-07], com 36 chamadas a função
```

2.4 Nelder-Mead

In [57]:

```
initial_simplex = np.zeros((4,3))

#pontos ramdomicos para o tetraedro
initial_simplex2 = np.array([
    [1, 2, 10],
    [2, 8, 7],
    [3, 6, 9],
    [1, 5, 10],
])

initial_simplex3 = np.ones((4,3))


initial_simplex3 = np.ones((4,3))

for i_simplex in [initial_simplex, initial_simplex2, initial_simplex3]:
    minimize(function_param, initial_values, method="Nelder-Mead", tol=tol,
             options={'disp':True, 'initial_simplex':i_simplex, 'xatol': tol})
```

```
Optimization terminated successfully.
         Current function value: 2.000000
         Iterations: 1
         Function evaluations: 4
Optimization terminated successfully.
         Current function value: -2619.950495
         Iterations: 120
         Function evaluations: 215
Optimization terminated successfully.
         Current function value: -0.000000
         Iterations: 1
         Function evaluations: 4
```

2.5 NEWUOA ou BOBYQA

In [58]:

```
#Usando bobyqa
lower = np.array([-10.0, -10.0, -10.0])
upper = np.array([10, 10, 10])

soln = pybobyqa.solve(function_param, initial_values,
                      bounds=(lower,upper), seek_global_minimum=True)
print(soln)
```

```
****** Py-BOBYQA Results ******
Solution xmin = [ 1.00000000e+01  4.55445549e-01 -9.05772580e-09]
Objective value f(xmin) = -938.950495
Needed 400 objective evaluations (at 400 points)
Did a total of 11 runs
Approximate gradient = [-8.20000308e+01 -2.29299040e-05  3.42943321e
-05]
Approximate Hessian = [[ -533.43648773   497.29519866   131.5039109
7]
 [  497.29519866  1945.56614343 -1528.94559698]
 [  131.50391097 -1528.94559698  2504.85776728]]
Exit flag = 1
Warning (max evals): Objective has been called MAXFUN times
******************************
```

3.6 Descida do gradiente implementado com TensorFlow

In [59]:

```
x1 = tf.Variable(0.5, name='x1', dtype=tf.float32)
x2 = tf.Variable(0.5, name='x2', dtype=tf.float32)
x3 = tf.Variable(0.5, name='x3', dtype=tf.float32)

# Não usei as derivadas aqui
dx1_tf = tf.constant(-102, dtype=tf.float32) + tf.square(x1)
dx2_tf = tf.constant(-92, dtype=tf.float32) + tf.multiply(tf.constant(202, dtype
=tf.float32), x2)
dx3_tf = tf.multiply(tf.constant(108, dtype=tf.float32), x3)

#Usei só a função.
fun_tf = tf.multiply(tf.constant(-100, dtype=tf.float32), (x1 - tf.square(x2)))
          + tf.square(x1-tf.constant(1, dtype=tf.float32))
          - tf.multiply(tf.constant(90, dtype=tf.float32), x2 - tf.square(x3))
          + tf.square(x2 - tf.constant(1, dtype=tf.float32))


optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(fun_
tf)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    print("x's:", sess.run([x1,x2,x3]), " - F(x1,x2,x3):", sess.run(fun_tf))

    for i in range(10000):
        sess.run(optimizer)

    print( "End: x's:", sess.run([x1,x2,x3]), " - F(x1,x2,x3):", sess.run(fun_tf
))
```

```
x's: [0.5, 0.5, 0.5]  - F(x1,x2,x3): -47.0
End: x's: [50.99905, 0.4554456, 0.0]  - F(x1,x2,x3): -2619.95
```