ASSIGNMENT 2: Huber Loss

Rubén Martínez Sánchez de la Torre

1. **Please use the linear model notebook as a starting point.**
2. **Modify the data generating process by adding extra noise that would generate outliers. In the notebook for in-class instruction I used 4*binomial(1, 0.03, size=n) - 4*binomial(1, 0.03, size=n). Feel free to use any other distribution that generates outliers. (Please keep track of which data points contain this extra noise and which don't.)**

```
beta_true = 6.0
alpha_true = 3.0
n = 30
x = random(n)
noise_p = 4*np.random.binomial(1, 0.03, size=n)
print("\nnoise_p: " + str(noise_p))
noise_n = -4*np.random.binomial(1, 0.03, size=n)
print("\nnoise_n: " + str(noise_n))
y = linear_function(alpha_true,beta_true,x)+0.2*randn(n) + noise_p + noise_n
print("\nx: " + str(x))
print("\ny: " + str(y))
index_without_noise = []
for i in range(n):
    if noise_p[i] == 0 and noise_n[i] == 0:
        index_without_noise.append(i)
print("\nindex_without_noise: " + str(index_without_noise))
l_x_wo_n = []
l_y_wo_n = []
for e in index_without_noise:
    l_x_wo_n.append(x[e])
    l_y_wo_n.append(y[e])
x_wo_noise = np.array(l_x_wo_n)
y_wo_noise = np.array(l_y_wo_n)
print("\nx_wo_noise: " + str(x_wo_noise))
print("\ny_wo_noise: " + str(y_wo_noise))

beta = -1.
alpha = 3.
beta_mse = -1.
alpha_mse = 3.
learning_rate = 0.1
delta = 0.9
learning_rate = 0.1
delta = 1.5
```

```
noise_p: [0 0 0 0 0 0 0 0 0 0 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

noise_n: [ 0  0  0  0  0  0 -4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0 -4]

x: [0.29399978 0.4129318  0.05961663 0.29954612 0.13843366 0.90874738
 0.34375887 0.49904251 0.39948477 0.79519863 0.72101645 0.97850997
 0.75716427 0.08716617 0.61638641 0.17962276 0.17602176 0.66321435
 0.03889551 0.91619702 0.25032025 0.25791853 0.14376305 0.65457381
 0.1340307  0.76898842 0.31247384 0.58766446 0.45509084 0.71167376]

y: [ 4.69036016  5.71659992  3.48120526  4.80784106  4.04941554  8.32905152
  1.12241502  5.83369857  5.51268395  7.69947881  7.49774962 13.30593258
 11.81152026  3.43757165  6.6649355   4.19377295  4.0108329   6.79651494
  3.06838783  8.76671338  4.51662212  4.46578936  3.81735368  6.82837929
  3.38152965  7.61339938  4.92528444  6.41906696  5.86820171  3.59871167]

index_without_noise: [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]

x_wo_noise: [0.29399978 0.4129318  0.05961663 0.29954612 0.13843366 0.90874738
 0.49904251 0.39948477 0.79519863 0.72101645 0.08716617 0.61638641
 0.17962276 0.17602176 0.66321435 0.03889551 0.91619702 0.25032025
 0.25791853 0.14376305 0.65457381 0.1340307  0.76898842 0.31247384
 0.58766446 0.45509084]

y_wo_noise: [4.69036016 5.71659992 3.48120526 4.80784106 4.04941554 8.32905152
 5.83369857 5.51268395 7.69947881 7.49774962 3.43757165 6.6649355
 4.19377295 4.0108329  6.79651494 3.06838783 8.76671338 4.51662212
 4.46578936 3.81735368 6.82837929 3.38152965 7.61339938 4.92528444
 6.41906696 5.86820171]
```

3. **Instead of mean-squared-error loss, consider Huber loss with a reasonably chosen transition point between the quadratic and the linear part of the Huber loss function.**

```python
def derivatives_linear_part_huber(y_predicted):
    """
    loss = delta * (|y - (alpha + beta * x)|) - 0.5 * delta**2
    if |y - (alpha + beta * x)| > 0:
        loss = delta * (y - (alpha + beta * x)) - 0.5 * delta**2
        derivative loss wrt alpha = delta * (0 -(1 + 0)) - 0 = -delta
        derivative loss wrt beta = delta * (0 - (0 + x)) - 0 = -delta * x
    if |y - (alpha + beta * x)| < 0:
        loss = delta * (-1) * (y - (alpha + beta * x)) - 0.5 * delta**2
        derivative loss wrt alpha = delta * (-1) * (0 -(1 + 0)) - 0 = delta
        derivative loss wrt beta = delta * (-1) * (0 - (0 + x)) - 0 = delta * x
    Combining these 2 cases ( >0 and <0):
    """
    derivative_of_loss_wrt_alpha = - delta * np.sign(y - y_predicted)
    derivative_of_loss_wrt_beta = - delta * np.sign(y - y_predicted)
    return derivative_of_loss_wrt_alpha, derivative_of_loss_wrt_beta

def derivatives_quadratic_part_huber(y_predicted):
    # loss = 0.5 * (y - (alpha + beta * x))**2
    # derivative loss wrt alpha = -1 * (2 / 2) * (y - (alpha + beta * x))
    derivative_of_loss_wrt_alpha = -1*(y - y_predicted)
    # derivative loss wrt beta = -x * (2 / 2) * (y - (alpha + beta * x))
    derivative_of_loss_wrt_beta = -1*x*(y- y_predicted)
    return derivative_of_loss_wrt_alpha, derivative_of_loss_wrt_beta

# Function to update parameters (alpha and beta) for the model which uses Huber Loss
def update_alpha_and_beta_huber():
    global alpha, beta
    y_predicted = linear_function(alpha,beta,x)
    derivative_of_loss_wrt_alpha, derivative_of_loss_wrt_beta = np.where(np.abs(y - y_predicted) <= delta, derivatives_quadratic_part_huber(y_predicted), derivatives_linear_part_h
    alpha = alpha - learning_rate*derivative_of_loss_wrt_alpha.mean()
    beta = beta - learning_rate*derivative_of_loss_wrt_beta.mean()
    return alpha, beta

def huber_loss(y_pred, y, delta):
    error = y_pred - y
    is_smaller = np.abs(error) <= delta
    mse = np.square(error)
    mae = np.abs(error)
    return np.where(is_smaller, 0.5*mse, delta*(mae - 0.5*delta))
```

We can find the code to compute the derivative of loss function with respect to alpha and beta inside the “update_alpha_beta_huber” function but this precise line is cut in the previous image. The whole sentence is:

derivative_of_loss_wrt_alpha, derivative_of_loss_wrt_beta = np.where(np.abs(y - y_predicted) <= delta, derivatives_quadratic_part_huber(y_predicted), derivatives_linear_part_huber(y_predicted))

4. **Fit a regression line (or curve) through your data.**

   def animate(i):

   x = np.linspace(0,1,100)

   y = linear_function(alpha,beta,x)

   line.set_data(x,y)

   for i in range(20):

   a, b = update_alpha_and_beta_huber()

5. **Evaluate the mean-squared-error loss for the set of points that were not affected by the extra (outlier-generating) noise.**

   # Compute y_predicted_final for the case of checking the MSE of the points without extra noise using Huber Loss

   y_predicted_final_huber_wo_noise = linear_function(a, b, x_wo_noise)

   mse_huber_wo_noise = (np.square(y_predicted_final_huber_wo_noise – y_wo_noise)).mean()

   print("MSE Linear Function with SGD Huber Loss of data without noise: " + str(mse_huber_wo_noise))

6. **(6) Now evaluate the same (i.e. mean-squared-error loss for the set of points that were not affected by the extra outlier-generating noise) but for a standard regression line (or regression curve). Make sure you use the same data as in part 5.**

   def animate(i):

   x = np.linspace(0,1,100)

   y_std_reg = linear_function(alpha_mse, beta_mse,x)

   line_std_reg.set_data(x, y_std_reg)

   for i in range(20):

   c, d = update_alpha_and_beta_mse_std_reg_wo_noise()

   # Compute y_predicted_final for the case of standard regression using the points without extra noise

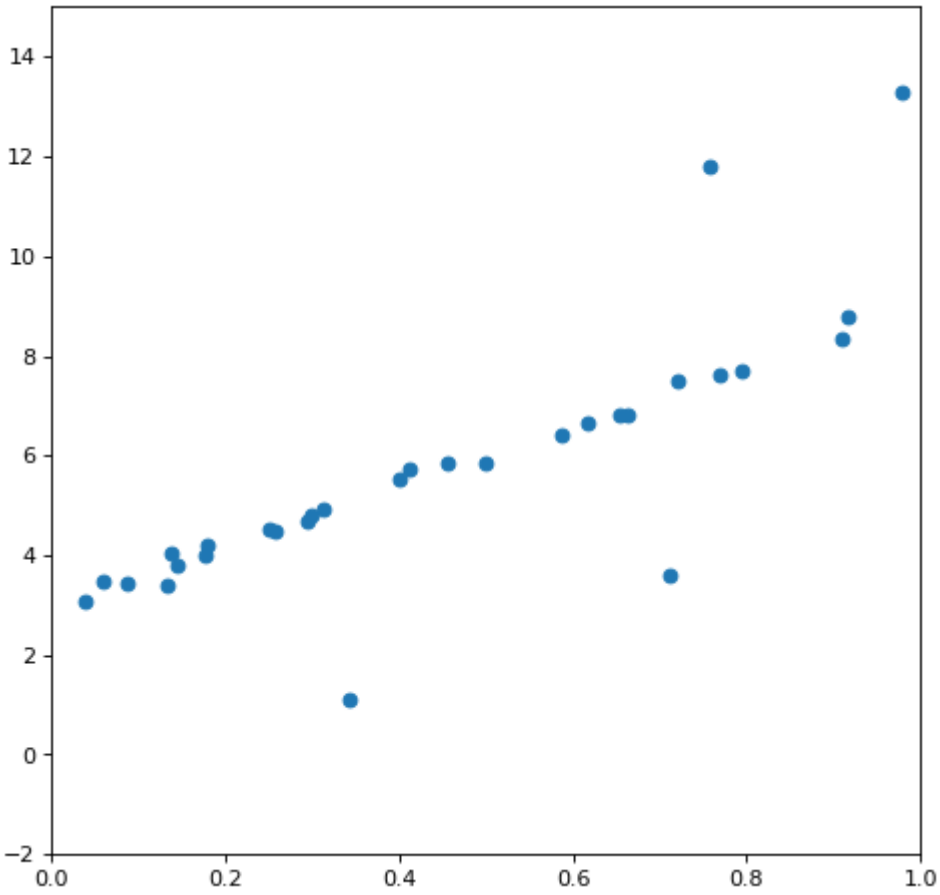
   y_predicted_final_mse = linear_function(c, d, x_wo_noise)

mse_wo_noise = (np.square(y_predicted_final_mse – y_wo_noise)).mean()

print("MSE Linear Function with SGD for Standard Regression of data without noise: " + str(mse_wo_noise))

7. **Find out which of these loss values is larger. Did Huber loss improve the result relative to standard regression?**

- First of all we're going to show the entire dataset:



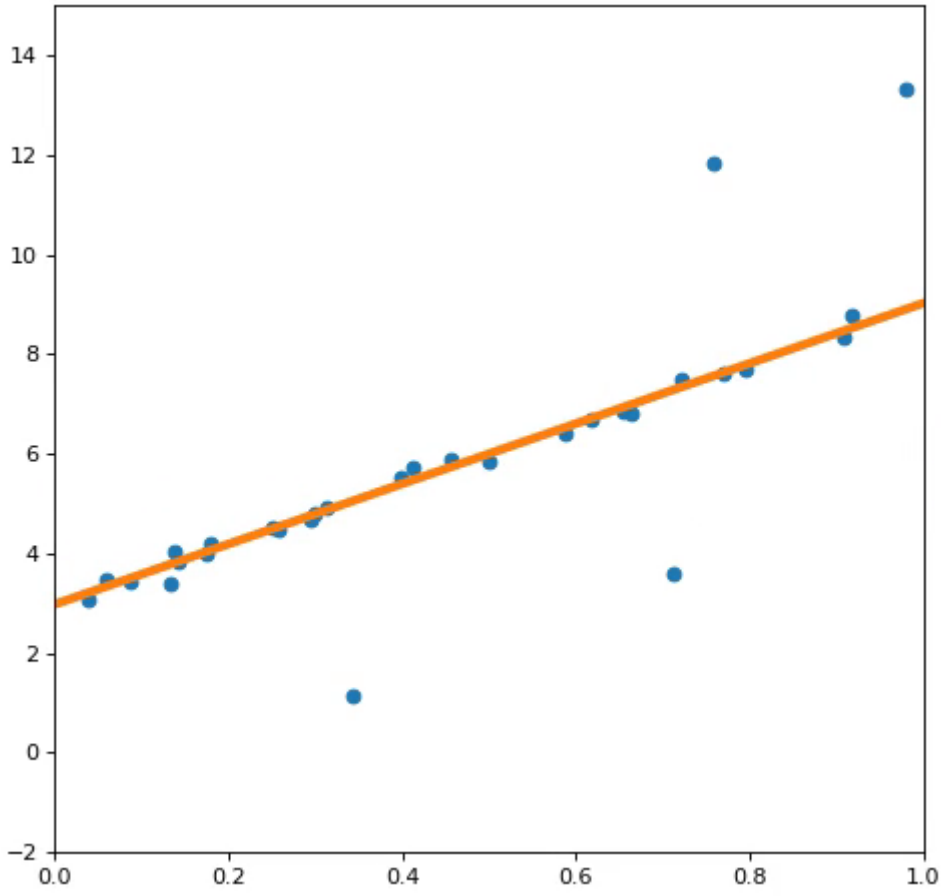
- I have printed the evolution of the MSE values for the required cases using the data without outliers ("MSE Linear Function with SGD Huber Loss of data without noise" , "MSE Linear Function with SGD for Standard Regression of data without noise") and the Huber loss values ("Huber Loss Linear Function with SGD") for the whole dataset. In the following image we can see the initial values:

```
Huber Loss Linear Function with SGD: [1.19085571 1.21285832 1.23486092 1.25686353 1.27886613 1.30086874
 1.32287134 1.34487395 1.36687655 1.38887916 1.41088177 1.43288437
 1.45488698 1.47688958 1.49889219 1.52089479 1.5428974  1.5649
 1.58690261 1.60890521 1.63090782 1.65291043 1.67491303 1.69691564
 1.71891824 1.74092085 1.76292345 1.78492606 1.80692866 1.82893127
 1.85093387 1.87293648 1.89493909 1.91694169 1.9389443  1.9609469
 1.98294951 2.00495211 2.02695472 2.04895732 2.07095993 2.09296254
 2.11496514 2.13696775 2.15897035 2.18097296 2.20297556 2.22497817
 2.24698077 2.26898338 2.29098598 2.31298859 2.3349912  2.3569938
 2.37899641 2.40099901 2.42300162 2.44500422 2.46700683 2.48900943
 2.51101204 2.53301464 2.55501725 2.57701986 2.59902246 2.62102507
 2.64302767 2.66503028 2.68703288 2.70903549 2.73103809 2.7530407
 2.7750433  2.79704591 2.81904852 2.84105112 2.86305373 2.88505633
 2.90705894 2.92906154 2.95106415 2.97306675 2.99506936 3.01707196
 3.03907457 3.06107718 3.08307978 3.10508239 3.12708499 3.1490876
 3.1710902  3.19309281 3.21509541 3.23709802 3.25910063 3.28110323
 3.30310584 3.32510844 3.34711105 3.36911365]
MSE Linear Function with SGD Huber Loss of data without noise: 2.8309810901097765
MSE Linear Function with SGD for Standard Regression of data without noise: 1.581232265252394
Huber Loss Linear Function with SGD: [0.05303085 0.05508452 0.05717721 0.0593089  0.06147961 0.06368934
 0.06593808 0.06822583 0.07055259 0.07291837 0.07532316 0.07776697
 0.08024978 0.08277162 0.08533246 0.08793232 0.09057119 0.09324908
 0.09596597 0.09872189 0.10151681 0.10435075 0.1072237  0.11013567
 0.11308664 0.11607664 0.11910564 0.12217366 0.12528069 0.12842674
 0.1316118  0.13483587 0.13809895 0.14140105 0.14474217 0.14812229
 0.15154143 0.15499958 0.15849675 0.16203293 0.16560812 0.16922233
 0.17287555 0.17656778 0.18029902 0.18406928 0.18787856 0.19172684
 0.19561414 0.19954045 0.20350578 0.20751012 0.21155347 0.21563584
 0.21975722 0.22391761 0.22811702 0.23235544 0.23663287 0.24094932
 0.24530478 0.24969925 0.25413274 0.25860524 0.26311675 0.26766728
 0.27225682 0.27688537 0.28155294 0.28625952 0.29100511 0.29578972
 0.30061334 0.30547597 0.31037762 0.31531828 0.32029796 0.32531664
 0.33037434 0.33547106 0.34060679 0.34578153 0.35099528 0.35624805
 0.36153983 0.36687062 0.37224043 0.37764925 0.38309709 0.38858393
 0.39410979 0.39967467 0.40527856 0.41092146 0.41660337 0.4223243
 0.42808424 0.4338832  0.43972116 0.44559815]
MSE Linear Function with SGD Huber Loss of data without noise: 1.8325537876377753
MSE Linear Function with SGD for Standard Regression of data without noise: 0.9794840140191953
```

- The final values can be checked in this screen shot:

```
Huber Loss Linear Function with SGD: [7.16721153e-25 6.84135281e-25 6.52814841e-25 6.21237824e-25
 5.91409119e-25 5.61843401e-25 5.33035779e-25 5.05432648e-25
 4.78128989e-25 4.51583425e-25 4.25795957e-25 4.00766583e-25
 3.76880763e-25 3.53355353e-25 3.30949238e-25 3.08578819e-25
 2.87327695e-25 2.67808811e-25 2.48037196e-25 2.28422958e-25
 2.10191988e-25 1.92719114e-25 1.76531688e-25 1.60550549e-25
 1.45327506e-25 1.30862558e-25 1.17155705e-25 1.04612817e-25
 9.23977001e-26 8.05837192e-26 6.99092479e-26 5.99928718e-26
 5.11181867e-26 4.24344058e-26 3.50270019e-26 2.81185525e-26
 2.19681985e-26 1.67380507e-26 1.20794326e-26 8.17890987e-27
 5.03648245e-27 2.65215036e-27 1.06653994e-27 1.57772181e-28
 4.77260848e-29 6.63037591e-28 2.10191988e-27 4.18451267e-27
 7.18849500e-27 1.08689256e-26 1.51524403e-26 2.03246068e-26
 2.62548686e-26 3.29432258e-26 4.03896783e-26 4.83177304e-26
 5.75568694e-26 6.69522027e-26 7.70575109e-26 8.86190564e-26
 1.00191646e-25 1.13318291e-25 1.26358162e-25 1.41049908e-25
 1.55557060e-25 1.71293651e-25 1.87788338e-25 2.05041121e-25
 2.23051999e-25 2.41820972e-25 2.61348040e-25 2.80967011e-25
 3.01985815e-25 3.23762715e-25 3.46297710e-25 3.69590800e-25
 3.93641986e-25 4.18451267e-25 4.44018644e-25 4.69483074e-25
 4.97427682e-25 5.24359394e-25 5.52934697e-25 5.83226955e-25
 6.10395325e-25 6.42202197e-25 6.74816862e-25 7.06126751e-25
 7.38146615e-25 7.73083687e-25 8.06570833e-25 8.40767953e-25
 8.78027431e-25 9.16094703e-25 9.52516411e-25 9.89648094e-25
 1.03258895e-24 1.06863674e-24 1.11059151e-24 1.15335409e-24]
MSE Linear Function with SGD Huber Loss of data without noise: 0.022849720536649295
MSE Linear Function with SGD for Standard Regression of data without noise: 0.022849720536649316
```

- Finally, this image shows how the orange line corresponding to the standard regression computed using the data without outliers and the blue line (which is not visible because of the orange line) corresponding to the Huber loss regression computed using the whole dataset fit the original data (blue dots).



- The intuition behind the Huber Loss function is based on using MSE or MAE when they can be more useful. The Mean Absolute Error (MAE) has the problem that usually leads to large gradients and this situation could avoid to find the minimum. In the other hand, the Mean Squared Error (MSE) gets smaller gradients as the minimum is getting closer but it also is sensitive to outliers.
- The Huber Loss function will have the following quadratic behavior: delta * (y – y_predicted)**2 when |y – y_predicted| < delta.
- And it will show this linear behavior: delta * (|y – y_predicted| - 0.5 * delta) when |y – y_predicted| > delta.
- Therefore the chose of delta will determine the behavior of the Huber Loss function and what data are considered as outliers.
- If delta is a big value then the Huber Loss approaches MSE.
- If delta is a very small value (closer to 0) then the Huber Loss approaches MAE.

- Summarizing, when the error |y – y_predicted| is greater then delta, this error will be minimized using MAE which is more robust to outliers than the MSE. Otherwise, the error will be minimized using the MSE which gets smaller gradients as we get closer to the minimum.