NISARG UPADHYAYA - 19CS30031

Assignment 2 Q7

Link to colab:
https://colab.research.google.com/drive/1Y6vHTjSqHCzkYIVDYuDyfLoIy5A47_RC?usp=sharing

Part C.

```
x_actual is
[[ 0.18206778]
 [-0.05961008]
 [-0.09865126]
 [ 0.27462866]
 [-0.17444411]
 [-0.10956963]
 [ 0.24197735]
 [ 0.20608811]
 [ 0.0501177 ]
 [ 0.20157481]]

x_iterative is
[[ 0.15734119]
 [-0.03867506]
 [-0.09975877]
 [ 0.25350486]
 [-0.1517311 ]
 [-0.07056733]
 [ 0.24160092]
 [ 0.20082604]
 [ 0.03976644]
 [ 0.18355747]]

2 norm of (x_actual - x_iterative) after 100 iterations is
0.06319639702296903
```

As it can be seen the actual and iterative answers are quite close. It is further observed that on increasing the number of iterations to 1000 and 10000 the answers obtained are even closer. Hence we have verified numerically that the algorithm converges to the optimal answer.

```
2 norm of (x_actual - x_iterative) after 1000 iterations is
1.7637357710782216e-05
2 norm of (x_actual - x_iterative) after 10000 iterations is
3.331393603522946e-15
```

Parts A, B and D.

NISARG UPADHYAYA

19CS30031

7

As $k \to \infty$ we have

$$x^k \approx x^{k+1}$$

The eq^n then takes the form

$$A^T(Ax^k - b) = 0$$

$$\Rightarrow A^T A x^k = A^T b$$

which is the normal eq^n.

Hence we have $x^k = \hat{x}$.

Notice that the eq^n given is a form of gradient descent.

We are trying to minimize $\|Ax-b\|_2^2$

$$= (Ax-b)^T(Ax-b)$$

It can be shown $\frac{\partial}{\partial x}(Ax-b)^T(Ax-b) = 2A^T(Ax-b)$.

Now we can write gradient descent as

$$x^{k+1} = x^k - \alpha\, 2A^T(Ax-b)$$

Taking the constant $\alpha = \dfrac{1}{2\|A\|^2}$ we get the given eq^n.

TIME COMPLEXITY

The multiplication Ax takes $O(mn)$ time.
The subtraction $Ax-b$ takes $O(m)$ time
The multiplication $A^T(Ax-b)$ takes $O(mn)$ time
The division with $||A||^2$ takes $O(m)$ time
The subtraction from x^k takes $O(m)$ time.
Hence total time is $O(mn)$.

Assuming this is run for 'k' steps we have time as $\underline{O(mnk)}$.

However, the time for calculating $||A||^2$ has not been taken into consideration. We can actually calculate it once at the beginning and reuse it at each step.

(d) The manual method requires QR factorization which is an expensive step and can takes around $O(mn^2)$ time. For a large number of features this might be very slow. The advantage of iterative method is the control over the number of iterations and we can stop early if successive x^k values are near same.

CODE

```
def iterative_least_squares(iter):
 A = np.random.rand(30,10)
 b = np.random.rand(30,1)
 rank = np.linalg.matrix_rank(A)
 print(f"Rank of A is: {rank}")
 if rank==10:
   print("A is full rank")
   x = np.zeros((10,1))
   x_actual = np.linalg.inv(A.T @ A) @ A.T @ b
   norm = np.linalg.norm(A,2)
   for i in range(iter):
       x_temp = x - ((A.T @ ((A @ x) - b)) / np.square(norm))
       x = x_temp
   print(f"x_actual is {x_actual}")
   print(f"x_iterative is {x}")
   print(f"2 norm of (x_actual - x_iterative) after {iter} iterations is
{np.linalg.norm(x - x_actual, 2)}")
 else:
   print("A is not full rank")
```