Ashutosh Kumar Singh 19CS30008

7. Iterative LS

(a) $$x^{(K+1)} = x^{(K)} - \frac{1}{\|A\|^2} A^T(Ax^{(K)} - b) \quad \text{--- (1)}$$

When the sequence $\{x^{(K)}\}$ converges, we can say that $x^{(K+1)}$ and $x^{(K)}$ become nearly equal.

$\Rightarrow x^{(K+1)} = x^{(K)}$

Hence, from (1),

$$\frac{1}{\|A\|^2} A^T(Ax^{(K)} - b) = 0$$

$$\Rightarrow A^T(Ax^{(K)} - b) = 0$$

$$\Rightarrow A^T A x^{(K)} = A^T b.$$

Thus, $x^{(K)}$ becomes a solution to the normal equation. Hence, we can say that $x^{(K)} = \hat{x}$, as $K \to \infty$.

(b) Computational Complexity:-

Computing $Ax^{(K)}$ $A \in R^{m \times n}$, $x^{(K)} \in R^n$.

$$Ax^{(K)} = x_1 \begin{bmatrix} A_1 \end{bmatrix} + x_2 \begin{bmatrix} A_2 \end{bmatrix} + \cdots + x_n \begin{bmatrix} A_n \end{bmatrix}$$

Computing each $x_i A_i$ involves m multiplications and there are n such terms, so total multiplications $= mn$.

Then, the number of total additions is also nearly mn.

Hence computing $Ax^{(K)}$ takes $O(mn)$ time.

Now, we need to subtract $b \in R^m$ from $Ax^{(k)}$.
This involves m operations.

Now, we have $(Ax^{(k)} - b) \in R^m$.

We need to multiply A^T with these.

Similar to computing $Ax^{(k)}$, we can also argue that multiplying A^T with $(Ax^{(k)} - b)$ involves mn operations.

Now we have a vector in R^n :- $A^T(Ax^{(k)} - b)$.

Dividing it by $\|A\|^2$ takes n operations.

Finally computing $x^{(k)} - \frac{1}{\|A\|^2} A^T(Ax^{(k)} - b)$ involves n operations again.

So, total no. of operations =

$$mn + mn + m + mn + n + n$$

$$= 3mn + m + 2n \text{ operations}$$

$$= O(mn) \text{ complexity.}$$

This the complexity for one iteration, for k iterations the computational complexity will be $O(mnk)$.

iterative_ls

October 21, 2021

[32]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

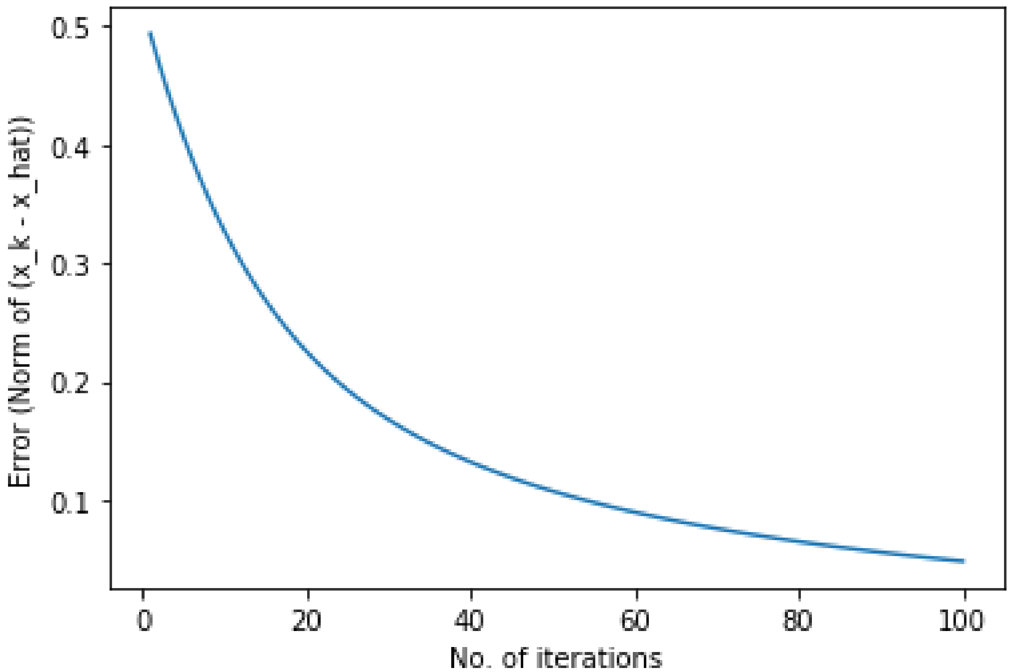
[33]:
```python
def iterative_ls(A, b, num_iter=100):
    iter_array = []
    error_array = []
    (m, n) = A.shape
    x = np.zeros(shape=(n,))

    # actual least squares solution
    x_hat = np.linalg.lstsq(A, b, rcond=None)[0]
    A_norm = np.linalg.norm(A, 2)

    # computing the least squares solution using the iterative method
    for i in range(num_iter):
        x = x - (1 / A_norm ** 2) * np.dot(np.transpose(A), (np.dot(A, x) - b))
        iter_array.append(i + 1)
        error_array.append(np.linalg.norm(x - x_hat))

    print(f'\nNo. of iterations: {num_iter}')
    print('\nLeast squares solution computed using iterative method:\n', x)
    print('\nActual least squares solution:\n', x_hat)
    print()

    plt.plot(iter_array, error_array)
    plt.xlabel('No. of iterations')
    plt.ylabel('Error (Norm of (x_k - x_hat))')
    plt.show()
```

[34]:
```python
def main():
    m, n = 30, 10
    A = np.random.rand(m, n)
    b = np.random.rand(m,)
    print('\nMatrix A:\n', A)
    print('\nVector b:\n', b)
    print(f'\nRank of A = {np.linalg.matrix_rank(A)}')
    iterative_ls(A, b)
```

[35]:
```
if __name__ == '__main__':
    main()
```

```
Matrix A:
 [[0.10484196 0.49127509 0.29205227 0.77837329 0.97869137 0.03686584 0.93810612
0.42499212 0.80057911 0.18858136]
 [0.02053662 0.41492819 0.34005467 0.79269661 0.74204085 0.46347802 0.37305066
0.26959222 0.11943535 0.16228338]
 [0.91014035 0.43827127 0.0763001  0.36234303 0.33015128 0.98151986 0.33183244
0.62686524 0.68228988 0.07409799]
 [0.20661736 0.64479052 0.66886456 0.41258828 0.69135937 0.71859448 0.38595877
0.66332177 0.31743178 0.21765278]
 [0.09624722 0.96603107 0.90736335 0.59597628 0.05239256 0.06842924 0.85349358
0.6990773  0.55906966 0.48450471]
 [0.23643858 0.97288492 0.31164037 0.21468694 0.16069021 0.62162248 0.69652459
0.32143516 0.76384143 0.30166032]
 [0.72453581 0.69700943 0.75197426 0.97511366 0.76117678 0.57763882 0.48308098
0.77568061 0.1804135  0.22494921]
 [0.48105023 0.06758852 0.67715448 0.76871693 0.39748284 0.60548351 0.14681578
0.0011338  0.28560471 0.8293663 ]
 [0.60562576 0.35151241 0.21128282 0.32149092 0.46490662 0.1730759  0.12867197
0.46126265 0.03882957 0.28906998]
 [0.75987536 0.92239133 0.33407418 0.35162488 0.90646224 0.85239003 0.34936526
0.24072192 0.78695771 0.56700918]
 [0.59215888 0.71521802 0.19702333 0.08328943 0.36167922 0.65741542 0.21567263
0.95260399 0.28351824 0.30147896]
 [0.72844805 0.56601404 0.37847532 0.98068974 0.55398544 0.77761771 0.00939231
0.90757092 0.00789554 0.50144053]
 [0.28175871 0.28711288 0.64207705 0.79089772 0.51664272 0.47091292 0.02305687
0.02894519 0.98004367 0.5498692 ]
 [0.25531656 0.43688319 0.46107014 0.8612016  0.27455376 0.75771064 0.86236655
0.61931255 0.77171139 0.61517508]
 [0.44839071 0.33361124 0.75710619 0.03908722 0.68682383 0.32355763 0.43133765
0.31559491 0.2110224  0.9725761 ]
 [0.6017269  0.8004521  0.38761572 0.75056553 0.58218495 0.14365484 0.82943102
0.58878764 0.3015226  0.97155061]
 [0.01956025 0.13468183 0.2836587  0.2266052  0.62853034 0.20142816 0.11386933
0.57498133 0.99145987 0.12341424]
 [0.84388797 0.94063662 0.96900195 0.34565014 0.73376425 0.8765702  0.37844971
0.794886   0.49448946 0.87940151]
 [0.28796504 0.05475747 0.62338553 0.54683269 0.34071377 0.18334494 0.29915149
0.34086699 0.84593491 0.95090026]
 [0.97187986 0.22859206 0.59310073 0.21558106 0.34380682 0.68201608 0.80031313
0.54089278 0.59617134 0.98019212]
 [0.97625332 0.65497646 0.11813555 0.12985878 0.28188693 0.5721979  0.12966113
0.86648566 0.75519911 0.57863785]
```

```
 [0.46680897 0.96065378 0.41185493 0.0839593  0.97614256 0.0693204  0.74816813
0.99128082 0.77657131 0.40313423]
 [0.63129328 0.79546845 0.54279789 0.10994432 0.18965693 0.88512612 0.04151282
0.20147169 0.06583035 0.12555145]
 [0.64529922 0.1395722  0.140746   0.855133   0.62280783 0.93268649 0.21770583
0.58926135 0.71227682 0.9617357 ]
 [0.23238416 0.8614355  0.37075709 0.1319121  0.47211542 0.48768538 0.6372917
0.85851143 0.12830248 0.75125762]
 [0.73875633 0.96913711 0.92724262 0.52709059 0.63854468 0.89695497 0.98714738
0.70232824 0.07852498 0.60244151]
 [0.58545662 0.01740384 0.45660554 0.88490276 0.79835637 0.6326668  0.51020435
0.03675152 0.62719088 0.08585902]
 [0.61486629 0.98589909 0.84140571 0.00257355 0.7300119  0.77607145 0.81723499
0.01664876 0.56110756 0.00750938]
 [0.11015196 0.87408556 0.76745738 0.4672763  0.4387354  0.63247089 0.92402249
0.65893606 0.02086774 0.77941783]
 [0.54270126 0.35896935 0.19480573 0.95949366 0.46112077 0.05737044 0.03591517
0.38692301 0.81039807 0.27541833]]

Vector b:
 [0.34794696 0.87011376 0.7115247  0.21723935 0.16036769 0.5731697  0.45413076
0.21949781 0.11514921 0.00274403 0.34460497 0.21226281 0.76635055 0.67405956
0.65463809 0.61227869 0.75148033 0.26024789 0.40493538 0.1598306  0.63898888
0.06448784 0.11543478 0.78004077 0.36260928 0.3526173  0.88735847 0.95757222
0.21394411 0.82540963]

Rank of A = 10

No. of iterations: 100

Least squares solution computed using iterative method:
 [ 0.0192908  -0.07494038 -0.0073631   0.25472518  0.16063701  0.22364411
0.09216539 -0.07164809  0.3871108  -0.09409771]

Actual least squares solution:
 [-0.00215346 -0.09281525 -0.01881895  0.24975333  0.15942945  0.25515922
0.11018624 -0.06297752  0.39036745 -0.09802825]
```



0.5
0.4
0.3
0.2
0.1
Error (Norm of (x_k - x_hat))
0
20
40
60
80
100
No. of iterations

(d) The direct ~~me~~ method for computing the least squares solution is:-

1. Compute the QR factorization:- $A = QR$
2. Compute $Q^T b$.
3. Solve $R\hat{x} = Q^T b$ using back-substitution.

Now, in the ~~me~~ iterative method, the only slightly expensive operations are multiplying vectors by A and A^T. So, if we have faster and efficient methods for calculating these matrix-vector products, then the iterative method may be computationally beneficial over the direct methods of the LS problem, as then we will be saved from incurring the computational cost of the QR factorization and the back substitution steps.