Problem Set 1

Name: Jianxiang Fan
Email: jianxiang.fan@colorado.edu

1.
$$B = (\mathbf{x}\mathbf{y}^T)^k = \mathbf{x}(\mathbf{y}^T\mathbf{x})^{k-1}\mathbf{y}^T = (\mathbf{y}^T\mathbf{x})^{k-1}\mathbf{x}\mathbf{y}^T$$

Power-of-Outer-Product($\mathbf{x}$, $\mathbf{y}$, k)

$a = 0$
for $i = 1$ to n
    $a = a + x_i y_i$
$b = a^{k-1}$
for $i = 1$ to n
    for $j = 1$ to n
        $B_{ij} = b\, x_i y_j$

Power-of-Scalar(a, k)

$b = 1$
while $k > 0$
    if $k \% 2 \neq 0$
        $b = b * a$
    $a = a * a$
    $k = k \gg 1$
return b

Time complexity (FLOPs) of Power-of-Outer-Product:

$$\begin{aligned} 2\,O(n) + 2\,O(\lg k) + 2\,O(n^2) &= O(n^2) + O(\lg k) \\ &= O(n^2) \text{ (If } k \ll n) \end{aligned}$$

2. Here is the implementation of axrow and axcol, and also the driver function.

```
function y = axrow(A, x)
    [n,~] = size(x);
    y = zeros(n,1);
    for i=1:n
        for j=1:n
            y(i,1) = y(i,1) + A(i,j)*x(j,1);
        end
```

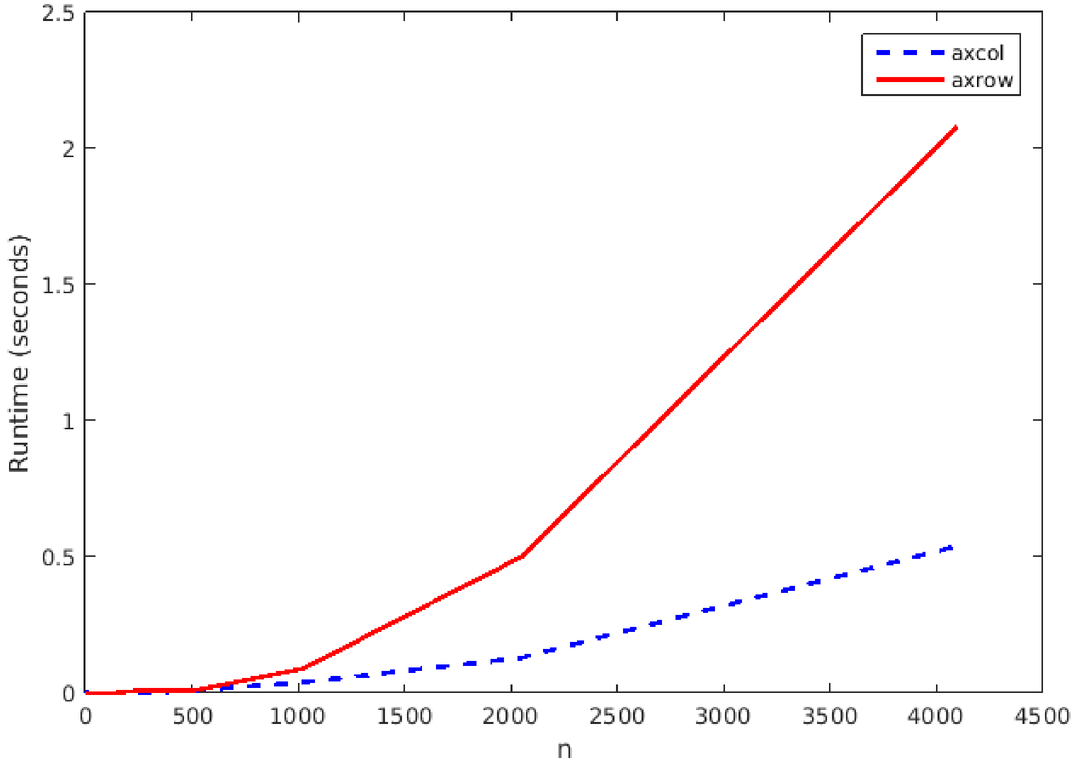
```
    end
end

function y = axcol(A, x)
    [n,~] = size(x);
    y = zeros(n,1);
    for j=1:n
        for i=1:n
            y(i,1) = y(i,1) + A(i,j)*x(j,1);
        end
    end
end

function run_test(limit)
    n = 1;
    time_col = zeros(1,limit);
    time_row = zeros(1,limit);
    for i=1:limit
        n = n * 2;
        A = rand(n);
        x = rand(n, 1);
        st = cputime;
        axrow(A,x);
        time_row(1,i) = cputime - st;
        st = cputime;
        axcol(A,x);
        time_col(1,i) = cputime - st;
    end
    x_plot = 2.^(1:limit);
    figure;
    plot(x_plot,time_col, 'b--', x_plot, time_row, 'r-','LineWidth', 2);
    legend('axcol','axrow');
    xlabel('n');
    ylabel('Runtime (seconds)');
end
```

Plot runtime with different n (size of the matrix). (See next page)

It shows that MATLAB stores matrices in a column-major (columnwise) scheme.

3. The naive implementation of matrix-matrix product may use triple-nested for-loops. For example:

```
C = zeros(n);
```




```
for i=1:n
    for j=1:n
        for k=1:n
            C(i,j) = C(i,j) + A(i,k)*B(k,j);
        end
    end
end
```

For the above version, we call the ordering of loop is “ijk”, from the outest to the innest. So all the six possible orders are ” “ijk”, “ikj”, “jik”, “jki”, “kij”, “kji”.

Plot their runtimes with different matrix sizes n. (See next page)

Fast: “kji” and “jki”. MATLAB stores matrices in a columnwise scheme. Suppose it will cache one column each time for each matrix, it means that when we visit $C(1,1)$, $C(1,1), C(2,1), \cdots, C(n,1)$ will be put into cache. So when “i” is iterated in the innest loop, most number of cache-hits achieved for all the three variables $C(i,j)$, $A(i,k)$ and $B(k,j)$.

Slow: “ijk” and “jik”. For these cases, “k” is in the innest loop, so caches hit for $C(i,j)$ and $B(k,j)$, but miss for $A(i,k)$.

Really slow: "ikj" and "kij". For these cases, "j" is in the innest loop, caches only hit for $A(i, k)$, but miss for both $C(i, j)$ and $B(k, j)$.