# MAT 226B Large Scale Matrix Computation
# Final Project

Ahmed Mahmoud

March, 22nd 2020

## Problem 1:

(a) We know from nonsymmetric Lanczos process that

$$MV_k = V_k T_k + \beta_{k+1}[0 \dots 0 v_{k+1}]$$

We can multiply the above by $e_1$ to extract the first column ($v_1$) from $V_k$ before multiplying it by $M$ and the result is

$$\begin{aligned} MV_k e_1 &= V_k T_k e_1 + \beta_{k+1}[0 \dots 0 v_{k+1}]e_1 \\ MV_k e_1 &= V_k T_k e_1 + 0 \end{aligned}$$

Note that $MV_k e_1 = Mv_1$. Now, we can easily give the proof as

$$M^j r = M^j(\beta_1 v_1) = \beta_1 M^j v_1$$

$$M^j r = \beta_1 V_k T_k^j e_1, \quad \forall j = 0, 1, \dots, k-1 \tag{1}$$

For the second part, we note that $e_k^T T_k^{k-1} e_1 = 0$. Thus, the second sum has no effect. We can let $j = k$ in 1 and we get

$$M^k r = \beta_1 V_k T_k^k e_1 + \beta_1 \beta_{k+1} (e_k^T T_k^{k-1} e_1) v_{k+1}$$

(b) We follow the same steps as in (a). First we have

$$
\begin{aligned}
M^T W_k &= W_k \hat{T}_k + \gamma_{k+1}[0 \ldots 0 w_{k+1}] \\
M^T W_k e_1 &= W_k \hat{T}_k e_1 + \gamma_{k+1}[0 \ldots 0 w_{k+1}] e_1 \\
M^T w_1 &= W_k \hat{T}_k e_1 + 0
\end{aligned}
$$

Taking the transpose of the above, we get

$$(M^T w_1)^T = w_1^T M = e_1^T \hat{T}_k{}^T W_k^T$$

We also know that $\hat{T}_k{}^T = D_k T_k D_k^{-1}$. Thus,

$$w_1^T M = e_1^T D_k T_k D_k^{-1} W_k^T$$

Now, we can give the proof as

$$
\begin{aligned}
c^T M^j &= \gamma_1 w_1^T M^j \\
c^T M^j &= \gamma_1 e_1^T D_k T_k^j D_k^{-1} W_k^T \\
c^T M^j &= \gamma_1 \delta_1 e_1^T T_k^j D_k^{-1} W_k^T
\end{aligned}
$$

(c) We can write the $Z(s)$ as

$$Z(s) = \sum_{j=0}^{\infty} \sigma^j c^T M^j r \tag{2}$$

We can find two values positive $j_1$ and $j_2$ such that $j_1 + j_2 = j$. Then, we can write 2 as

$$
\begin{aligned}
Z(s) &= \sum_{j=0}^{\infty} \sigma^j c^T M^{j_1} M^{j_2} r \\
Z(s) &= \sum_{j=0}^{\infty} \sigma^j (\gamma_1 \delta_1 e_1^T T_k^{j_1} D_k^{-1} W_k^T)(\beta_1 V_k T_k^{j_2} e_1)
\end{aligned}
$$

$$Z(s) = \sum_{j=0}^{\infty} \sigma^j \gamma_1 \delta_1 \beta_1 e_1^T T_k^{j_1} D_k^{-1} W_k^T V_k T_k^{j_2} e_1 \tag{3}$$

We know from Lanczos process that $W_k V_k = D_k$. In addition, we have $c^T r = (\gamma_1 w_1)^T(\beta_1 v_1) = \gamma_1\beta_1 w_1^T v_1 = \gamma_1\delta_1\beta_1$. We can plug this relations in 3 to get

$$Z(s) = \sum_{j=0}^{\infty} \sigma^j (c^T r) e_1^T T_k^{j_1} D_k^{-1} D_k T_k^{j_2} e_1$$
$$Z(s) = \sum_{j=0}^{\infty} \sigma^j (c^T r) e_1^T T_k^{j_1} T_k^{j_2} e_1 = \sum_{j=0}^{\infty} \sigma^j (c^T r) e_1^T T_k^{j} e_1$$

# Problem 2:

Here we are required to find an efficient way to compute $q = Mv$ and $q = M^T v$ for $v \in \mathbb{C}^n$ where $M = (A - s_0 E)^{-1} E$. We can compute the matrix-vector multiplication efficiently using LU factorization. We first can write the multiplication as

$$q = (A - s_0 E)^{-1} E v = (\underbrace{A - s_0 E}_{W})^{-1} \underbrace{Ev}_{f}$$
$$q = W^{-1} f \quad \Rightarrow \quad Wq = f \quad \Rightarrow \quad \underbrace{PD^{-1}WQ}_{LU} \underbrace{Q^T q}_{d} = PD^{-1} f$$

Thus, we can fist solve $Lc = PD^{-1}f$ for $c \in \mathbb{C}^n$ via forward substitution, then solve $Ud = c$ for $d \in \mathbb{C}^n$ via backward substitution, and finally set $q = Qd$.

We can use the same LU factorization to compute $q = M^T v$ efficiently. We first not that transposing the LU factorization for a given matrix $W$ is $U^T L^T = Q^T W^T D^{-T} P^T$ We can write this multiplication as

$$q = ((A - s_0 E)^{-1} E)^T v = E^T \underbrace{(A - s_0 E)^{-T} v}_{g}$$
$$g = W^{-T} v \quad \Rightarrow \quad W^T g = v \quad \Rightarrow \quad \underbrace{Q^T W^T D^{-T} P^T}_{U^T L^T} \underbrace{(D^{-T} P^T)^{-1} g}_{d} = Q^T v$$

Thus, we can first solve $U^T c = Q^T v$ for $c$ via forward substitution, then solve $L^T d = c$ for $d$ via backward substitution, and then set $g = D^{-T} P^T d$. Finally, we multiply $g$ from the left by $E^T$ to get $q$. The functions `Mv` and `transposeMv` implements these operations as discussed.

## Problem 3:

The leading $2k$ moments $\mu_j = c^T M^j r$ for $j = 0, 1, \ldots, 2k-1$ can be computing as follows. Let $f_j = M^j r$. It is easy to see that $f_j = M f_{j-1}$ from which we can compute the moment at $j$ as $\mu_j = c^T f_j$ and compute $f_j$ recursively. We can use the same LU factorization to compute $r$ and used the function `Mv` to compute $f_j$. The function `computeMoments` compute the moments as discussed here.

We wrote another function `textbookAlgo` that utilizes `computeMoments` to implement the textbook algorithm for computing $Z_k(s)$. More precisely, it compute the coefficient of the polynomials $p(\sigma)$ and $q(\sigma)$ such that $Z_k(s) = \frac{p(\sigma)}{q(\sigma)}$ where $p(\sigma) = \alpha_0 + \alpha_1\sigma + \cdots + \alpha_{k-1}\sigma^{k-1}$, $q(\sigma) = \beta_0 + \beta_1\sigma + \cdots + \beta_k\sigma^k$, $\alpha_0, \ldots, \alpha_{k-1}, \beta_1 \ldots \beta_k \in \mathbb{C}$, and $\beta_0 = 1$. The output of this function is two vectors $\alpha$ and $\beta$ containing the coefficients.

## Problem 4:

We wrote the function `zkViaLanczos` which computes $Z_k$ given $T_k$, $s$ and $s_0$. $T_k$ is computed from our previous implementation of the nonsymmetric Lanczos in Homework 3 which feed in with the efficient implementation of the $Mv$ and $M^T v$ from Problem 2.

## Problem 5:

**System Specs:** All our experiments run on Intel(R) Xeon(R) CPU E3-1280 v5 with 3.70 GHz and 32 GB of RAM on 64-bit operating system running Windows 7.

**Plots:** Figure 1 shows the results of the three algorithms plotted on top of each others. It shows that Lanczos-based algorithm is able to capture $Z(s)$ almost exactly using $k = 100$. For such value of $k$, the textbook algorithm will return NaN everywhere. Thus, we used $k = 10$ in the plot. Function `Figure_1()` in `driver.m` file generates this plot.

**$s_0$ with fast convergence:** We test our implementation of the textbook and Lanczos-based algorithm for different values of $s_0$ and found that it runs fairly fast for the small input given in `FP_Ex1.mat`; it takes less than a second even for large i.e., $k < 100$.

We followed the recommendation given in the lectures for how to pick $s_0$. We choose $s_0 = 1\text{e}5 + 2\pi i 5.5\text{e}8$.

**Comparison:** We run both our implementation for different values of $k$ and the above $s_0$ and compared between both. Table 1 shows the average different ($\|\cdot\|^2$) and the maximum (absolute) different between the two vectors containing the output of both algorithms for different values of $s$. Function `Table_1()` in `driver.m` file generates these data. We can see that when $k > 13$, the two algorithms will give difference numerical results.

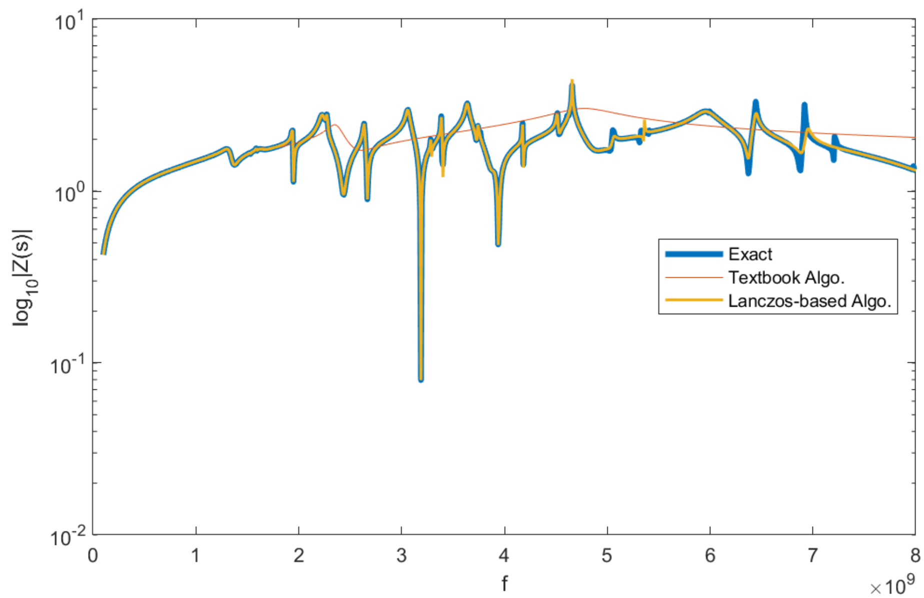


Figure 1: The results of the three algorithms; exact algorithm, textbook algorithm with $k = 10$, and Lanczos-based algorithm with $k = 100$. We used expansion point $s_0 = 1\text{e}5 + 2\pi i 5.5\text{e}8$ for both algorithms.

**Explanation:** We believe the reason why the textbook algorithm does not perform well is because it depends on computing $M^j r$ (to compute the moments) for increasing values of $j$ which convergences quickly to the the eigenvector of $M$ with largest eigenvalue. Thus, the information it contains comes from a single eigenvector where the information should comes from all eigenvectors of $M$. In contrast, Lanczos's $T_k$ represents oblique projection of $M$ onto the $K_k(M, r)$ Krylov subspace which contains information about $k$ eigenvectors.

**Lanczos approach with difference $s_0$:** For this experiment, we defined the "good approximation" such that the average difference between the Lanczos-based algorithm and the exact algorithm is less than $10^{-5}$. We tested using different $s_0$ and for each value we run the algorithm in a loop for $200 \leq k \leq 1000$ and stop when the results meet the good-approximation criterion we set thus obtaining the minimum $k$ value that results into the best approximation given $s_0$. Table 2 show the results for different $s_0$. Function `Table_2()` in `driver.m` file generates these data.

We notice that complex $s_0$ take more time for the same $k$ value (first and last row in Table 2. Expansion point with complex part equal to the maximum or minimum frequency take double the time it takes for $s_0$ suggested in the lecture notes. Getting closer y-axis can results in higher $k$ values and thus slower convergence.

| $k$ | Average Difference | Maximum Difference |
|---|---|---|
| 2 | 2.082747e−28 | 2.109424e−15 |
| 3 | 7.919421e−27 | 6.439294e−15 |
| 4 | 5.026520e−25 | 4.618528e−14 |
| 5 | 8.547583e−26 | 2.664535e−14 |
| 6 | 3.052289e−23 | 4.112266e−13 |
| 7 | 1.148403e−19 | 4.235057e−11 |
| 8 | 2.656897e−17 | 3.190033e−09 |
| 9 | 6.690131e−15 | 1.812676e−08 |
| 10 | 6.292776e−12 | 3.008865e−07 |
| 11 | 7.619719e−11 | 6.315981e−06 |
| 12 | 4.360106e−04 | 1.650155e−02 |
| 13 | 9.107709e−04 | 1.494378e−02 |
| 14 | 7.052574e+00 | 3.945073e−01 |
| 15 | 5.904627e+01 | 1.279984e+00 |
| 16 | 1.779754e+02 | 1.488481e+00 |
| 17 | 1.105689e+02 | 1.625567e+00 |
| 18 | 1.100795e+02 | 1.632208e+00 |
| 19 | 1.219843e+02 | 1.688382e+00 |
| 20 | 1.075413e+02 | 9.725151e−01 |
| 21 | 1.108709e+02 | 1.217680e+00 |
| 22 | 4.712087e+02 | 1.762950e+00 |
| 23 | 1.160591e+03 | 2.593543e+00 |
| 24 | 3.507356e+03 | 4.102138e+00 |
| 25 | 7.329648e+03 | 5.568756e+00 |
| 26 | 2.245490e+04 | 8.889336e+00 |
| 27 | 2.633420e+04 | 9.681861e+00 |
| 28 | 4.881768e+04 | 1.252447e+01 |
| 29 | 7.066637e+04 | 1.471937e+01 |
| 30 | 1.151561e+05 | 1.801101e+01 |

Table 1: Average and maximum (absolute) difference between the results of the textbook algorithm and Lanczos approach for different $k$ values.

| $s_0$ | $k$ | Time | Average Difference |
|---|---|---|---|
| $10^5 + 2\pi i f_{avg}$ | 212 | 3.416422 | 7.9513e−6 |
| $10^5 + 2\pi i f_{min}$ | 278 | 7.300847 | 6.419988e−6 |
| $10^5 + 2\pi i f_{max}$ | 262 | 6.130839 | 8.102979e−7 |
| $10^9$ | 290 | 6.739243 | 8.396701e−6 |
| $10^{10}$ | 212 | 2.901619 | 4.187281e−6 |

Table 2: Lanczos approach using different $k$ and $s_0$ values and comparing it with the exact solution ($f_{avg} = \frac{f_{min}+f_{max}}{2}$)