MAT 226B Large Scale Matrix Computation
Final Project

Ahmed Mahmoud

March, 22nd 2020

Problem 1:

(a) We know from nonsymmetric Lanczos process that

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

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

$$\begin{aligned}MV_ke_1 &= V_kT_ke_1 + \beta_{k+1}[0\dots 0v_{k+1}]e_1\\ MV_ke_1 &= V_kT_ke_1 + 0\end{aligned}$$

Note that $MV_ke_1 = Mv_1$. Now, we can easily give the proof as

$$M^jr = M^j(\beta_1v_1) = \beta_1M^jv_1$$

$$M^jr = \beta_1V_kT_k^je_1, \quad \forall j = 0, 1, \dots, k-1 \tag{1}$$

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

$$M^kr = \beta_1V_kT_k^ke_1 + \beta_1\beta_{k+1}(e_k^TT_k^{k-1}e_1)v_{k+1}$$

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

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

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

$$q = (A - s_0E)^{-1}Ev = (\underbrace{A - s_0E}_{W})^{-1}\underbrace{Ev}_{f}$$
$$q = W^{-1}f \quad \Rightarrow \quad Wq = f \quad \Rightarrow \quad \underbrace{PD^{-1}WQ}_{LU}\underbrace{Q^Tq}_{d} = PD^{-1}f$$

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

$$q = ((A - s_0E)^{-1}E)^T v = E^T \underbrace{(A - s_0E)^{-T}v}_{g}$$
$$g = W^{-T}v \quad \Rightarrow \quad W^Tg = v \quad \Rightarrow \quad \underbrace{Q^TW^TD^{-T}P^T}_{U^TL^T}\underbrace{(D^{-T}P^T)^{-1}g}_{d} = Q^Tv$$

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 α and β containing the coefficients.