MAS565 Numerical Analysis HW7

20218125 채지석

6.11. (a) Recall that $lub_2(A) = \max_{\|x\|_2=1} \|Ax\|_2$. Let $A=(a_{ij})$, then for any $x \in \mathbb{C}^n$ we have

$$\| |A| x \|_2^2 = \sum_{i=1}^n \left| \sum_{j=1}^n |a_{ij}| x_j \right|^2$$
$$\le \sum_{i=1}^n \left(\sum_{j=1}^n |a_{ij}||x_j| \right)^2$$
$$= \| |A| |x| \|_2^2$$

while $\| |x| \|_2^2 = \sum_{i=1} |x_i|^2 = \|x\|_2^2$. Therefore x^* with $\|x^*\|_2=1$ and $\||A|x^*\|_2 = lub_2(|A|)$ we may assume that $x \in \mathbb{R}^n$ and $x \ge 0$. Now, for any $x \ge 0$, if $|A| \le |B|$ let $B=(b_{ij})$ then

$$\| |A| x \|_2^2 = \sum_{i=1}^n \left(\sum_{j=1}^n |a_{ij}| x_j \right)^2 \le \sum_{i=1}^n \left(\sum_{j=1}^n |b_{ij}| x_j \right)^2 = \| |B| x \|_2^2$$

which also implies

$$lub_2(|A|) = \||A|x^*\|_2 \le \| |B| x^* \|_2 \le \sup_{\|x\|_2=1} \| |B| x \| = lub_2(|B|).$$

(b). For $A=(a_{ij})$ and $x \in \mathbb{C}^n$ we have

$$\|Ax\|_2^2 = \sum_{i=1}^n \left| \sum_{j=1}^n a_{ij} x_j \right|^2$$
$$\le \sum_{i=1}^n \left(\sum_{j=1}^n |a_{ij}||x_j| \right)^2$$
$$= \||A||x|\|_2^2 .$$

Let x_* be the vector such that $\|x_*\|_2 = 1$ and $\|Ax_*\|_2 = lub_2(A)$ then

$$lub_2(A) = \|Ax_*\|_2 \le \| |A| |x_*| \|_2 \le \sup_{\|x\|_2=1} \||A|x\|_2 = lub_2(|A|)$$

and the first inequality is shown. For the other side inequality first note that for any $x \in \mathbb{C}^n$ by Cauchy-Schwarz inequality

$$\left|\sum_{j=1}^{n} |a_{ij}||x_j|\right|^2 \leq \sum_{j=1}^{n} |a_{ij}|^2 \sum_{i=1}^{n} |x_j|^2$$
$$= \left(\sum_{j=1}^{n} |a_{ij}|^2\right) \|x\|_2^2$$

so whenever $\|x\|_2 = 1$ we have

$$\| |A||x| \|_2^2 = \sum_{i=1}^{n}\left(\sum_{j=1}^{n} |a_{ij}||x_j|\right)^2$$
$$\leq \sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2$$
$$= \|A\|_F^2$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Furthermore we have

$$\|A\|_F^2 = \sum_{i=1}^{n} \sigma_i^2 \leq n\, \sigma_{max}^2 = n(\mathrm{lub}_2(A))^2$$

where σ_i's are the singular values of A, and σ_{max} the maximum singular value. In total we have, whenever $\|x\|_2 = 1$, the inequality

$$\| |A||x| \|_2 \leq n\, \mathrm{lub}_2(A).$$

In (a) we have seen that the supremum of the left hand side over $\|x\|_2 = 1$ is $\mathrm{lub}_2(|A|)$, so we are done.

6.14. Let $t_k = A^k y_0$ for each $k = 0, 1, 2, \ldots$ and write $y_0 = \sum_{i=1}^{n} c_i x_i$. Then $t_k = A^k y_0 = A^k(c_1 x_1 + \cdots + c_n x_n) = c_1 \lambda_1^k x_1 + \cdots + c_n \lambda_n^k x_n$. Now, for arbitrary norm $\|\cdot\|$, we claim that $y_k = \alpha_k t_k$ for some $\alpha_k > 0$. We proceed by induction on k. When $k=0$ there is nothing to show. Meanwhile

$$y_{k+1} = \frac{Ay_k}{\|Ay_k\|} = \frac{A\alpha_k t_k}{\|Ay_k\|} = \frac{\alpha_k}{\|Ay_k\|} \cdot A(A^k y_0) = \frac{\alpha_k}{\|Ay_k\|} t_{k+1}$$

so set $\alpha_{k+1} = \frac{\alpha_k}{\|Ay_k\|}$ then we are done. This shows that, regardless of the choice of a norm, y_{k+1} is a unit norm vector with the same direction as t_{k+1}, but t_{k+1} is independent of the choice of a norm.

Now put $\beta_k = \alpha_k \|t_k\|_2$ and $u_k = \frac{t_k}{\|t_k\|_2}$ so that $y_k = \alpha_k t_k = \beta_k \frac{t_k}{\|t_k\|_2} = \beta_k u_k$. Then

$$q_{ki} = \frac{(Ay_k)_i}{(y_k)_i} = \frac{(A\beta_k u_k)_i}{(\beta_k u_k)_i} = \frac{(Au_k)_i}{(u_k)_i},$$

$$r_k = \frac{y_k^T A y_k}{y_k^T y_k} = \frac{\beta_k^2 u_k^T A u_k}{\beta_k^2 u_k^T u_k} = \frac{u_k^T A u_k}{u_k^T u_k}$$

hence we may without loss of generality assume that the norm we use is a 2-norm, as u_k is y_k when 2-norm is in use.

(a) Rewrite t_k into

$$\frac{t_k}{\lambda_1^k} = c_1 x_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k x_k.$$

Then $c_1 \neq 0$ as we had the assumption $x_1^T y_0 \neq 0$ and $x_i^T x_k = \delta_{ik}$. Also as $(\lambda_2/\lambda_1)^k \geq (\lambda_j/\lambda_1)^k$ for all $j = 2, 3, \ldots, n$ we have $\left(\frac{t_k}{\lambda_1^k}\right)_i = (c_1 x_1)_i\left(1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right)\right)$. As the observation in the previous paragraph show that we can replace y_k by $\frac{t_k}{\lambda_1^k}$ and

$$\frac{At_k}{\lambda_1^k} = c_1\lambda_1 x_1 + c_2\lambda_2 \cdot \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k \cdot \lambda_n x_k$$

so $\left(\frac{At_k}{\lambda_1^k}\right)_i = (c_1\lambda_1 x_1)_i\left(1 + O\left((\lambda_2/\lambda_1)^k\right)\right)$. We needed that $(x_1)_i \neq 0$ in order to make $\left(\frac{\lambda_2}{\lambda_1}\right)^k$ negligible with respect to $(x_1)_i$ when k is sufficiently large. Now,

$$q_{ki} = \frac{(At_k/\lambda_1^k)_i}{(t_k/\lambda_1^k)_i} = \frac{(c_1\lambda_1 x_1)_i}{(c_1 x_1)_i} \frac{1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right)}{1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right)}$$

$$= \lambda_1\left(1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right)\right)^2$$

$$= \lambda_1\left(1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right)\right).$$

(b) Here we replace y_k by u_k to get

$$r_k = \frac{u_k^T A u_k}{u_k^T u_k} = \frac{(A^k y_0)^T}{\|A^k y_0\|} A \frac{(A^k y_0)}{\|A^k y_0\|}$$
$$= \frac{y_0^T A^{2k+1} y_0}{y_0^T A^{2k} y_0} .$$

Meanwhile we also have

$$y_0^T A^k y_0 = (c_1 x_1 + \cdots + c_n x_n)^T (c_1 \lambda_1^k x_1 + \cdots + c_n \lambda_n^k x_n)$$
$$= \sum_{i,j=1}^{n} c_i c_j \lambda_j^k x_i^T x_j$$
$$= \sum_{i=1}^{n} c_i^2 \lambda_i^k$$

hence the Rayleigh quotient becomes

$$r_k = \frac{\sum_{i=1}^{n} c_i^2 \lambda_i^{2k+1}}{\sum_{i=1}^{n} c_i^2 \lambda_i^{2k}} .$$

Some computation leads to

$$\lambda_1 - r_k = \frac{\sum_{i=1}^{n} c_i^2 (\lambda_i^{2k} \lambda_1 - \lambda_i^{2k+1})}{\sum_{i=1}^{n} c_i^2 \lambda_i^{2k}}$$
$$= \frac{\sum_{i=2}^{n} c_i^2 \lambda_i^{2k} (\lambda_1 - \lambda_i)}{\sum_{i=1}^{n} c_i^2 \lambda_i^{2k}}$$
$$\leq \frac{\sum_{i=2}^{n} c_i^2 \lambda_i^{2k} \cdot \max_{2 \leq i \leq n} |\lambda_1 - \lambda_i|}{c_1^2 \lambda_1^{2k}}$$
$$= \max_{2 \leq i \leq n} |\lambda_1 - \lambda_i| \cdot \sum_{i=2}^{n} \left(\frac{c_i}{c_1}\right)^2 \cdot \left(\frac{\lambda_i}{\lambda_1}\right)^{2k}$$
$$\leq \left(\max_{2 \leq i \leq n} |\lambda_1| + |\lambda_i|\right) \cdot \max_{2 \leq i \leq n} \left(\frac{c_i}{c_1}\right)^2 \cdot \sum_{i=2}^{n} \left|\frac{\lambda_i}{\lambda_1}\right|^{2k}$$
$$\leq 2|\lambda_1| \cdot \max_{2 \leq i \leq n} \left(\frac{c_i}{c_1}\right)^2 \cdot \left(\frac{\lambda_2}{\lambda_1}\right)^{2k} .$$

Therefore $r_k = \lambda_1 \left(1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^{2k}\right)\right)$.

6.15. (a) From the Geršgorin circle theorem, there is one eigenvalue that is of distance at most one from 21, one that is at distance at most one from -9, and others of modulus less than or equal to 5. Therefore, letting $\lambda_1, \dots, \lambda_5$ the eigenvalues of A we can reindex them so that

$$22 \geq |\lambda_1| \geq 20 > 10 \geq |\lambda_2| \geq 8 \geq |\lambda_3| \geq |\lambda_4| \geq |\lambda_5|.$$

Hence the only thing we should show is that, letting $x_1, \dots, x_5$ the unit 2-norm eigenvectors of A each corresponding to $\lambda_1, \dots, \lambda_5$ respectively, it holds that $e_5^T x_1 \neq 0$. For the sake of contradiction assume that $e_5^T x_1 = 0$ so that

$$e_5 = c_2 x_2 + c_3 x_3 + c_4 x_4 + c_5 x_5.$$

Then first observe that $\|e_5\|_2^2 = c_2^2 + c_3^2 + c_4^2 + c_5^2 = 1$. Also we have

$$Ae_5 = c_2\lambda_2 x_2 + c_3\lambda_3 x_3 + c_4\lambda_4 x_4 + c_5\lambda_5 x_5.$$

hence $\|Ae_5\|_2^2 = c_2^2\lambda_2^2 + c_3^2\lambda_3^2 + c_4^2\lambda_4^2 + c_5^2\lambda_5^2 \leq (c_2^2 + c_3^2 + c_4^2 + c_5^2)\lambda_2^2 \leq 10$. However Ae_5 is the last column of A, hence its 2-norm is at least 21. this is absurd, so $e_5^T x_1 \neq 0$.

Now we can rely on the result of Exercise 14. Since

$$\frac{A^k e_5}{c_1 \lambda_1^k} = x_1 + \frac{c_2}{c_1} \cdot \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \dots + \frac{c_5}{c_1}\left(\frac{\lambda_5}{\lambda_1}\right)^k x_5$$

converges to x_1, with $\frac{A^k e_5}{c_1 \lambda_1^k}$ becoming y_k when normalized, and that $\|y_k\|_2 = 1 = \|x_1\|$, we conclude that the iteration with initial vector e_5 converges to x_1.

(b) According to Exercise 14 we have $r_k = \lambda_1(1 + O((\lambda_2/\lambda_1)^{2k}))$. In part (a) we have seen that $\frac{\lambda_2}{\lambda_1} \geq \frac{1}{2}$, so the error of r_{k+5} is at least $4^5 \approx 1024$ times better than that of r_k. That is, we expect at least a gain of three decimal digits.

Computer Assignment

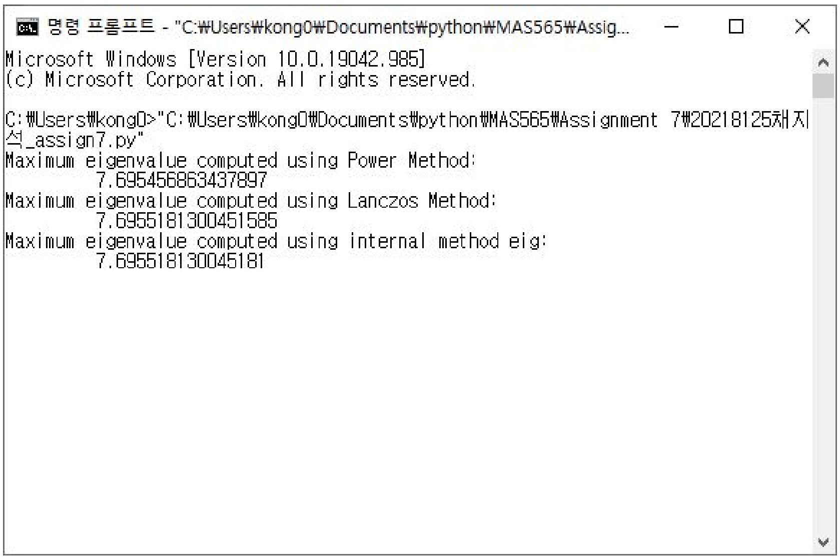
The program which does the required is submitted via KLMS along with this document. It performs two tasks, as required in the assignment sheet.

Firstly the program performs the Power method. The implementation details are straightforward, being a direct translation of the equations presented in Exercise 6.14. As in Assignment 5, the stop condition is set to be when the relative error is less than 10^{-6}.

After that the program performs the Lanczos method. In a broad sense the code is again a direct translation of equations (6.5.3.1a) and (6.5.3.1b) in the textbook. The iteration shall stop when a small value of $|\gamma_i|$ is met, and this threshold is again set to be 10^{-6}. Our goal is to compute the full tridiagonal matrix which is similar to A_7, but the iteration may stop early when the number of iterations has reached the dimension of the Krylov space $K(q, A_7)$ where q is the initial vector. When $q = e_1$ as indicated in the assignment sheet, this dimension turns out to be 25. This fact can be verifed by using the symbolic computation module `sympy` and the Python code included in the appendix. Anyhow, if the Lanczos iteration terminates early, this means that the tridiagonal matrix we desire is actually a block diagonal matrix of tridiagonal matrices, and we have to find a new initial vector to initiate the iteration in order to compute the next block. Such initial vector must be orthogonal to all q_i's computed up to that point, so we take the strategy of running through the canonical basis vectors and taking the orthogonal complement of the projections onto $\text{span}(q_i)$ until we find a nonzero orthogonal complement.

Once we obtain all γ_i's and δ_i's the rest is simple: we pass the computed tridiagonal matrix into the function `numpy.linalg.eig` provided by the `numpy` package in order to compute the eigenvalues.

The computed maximum eigenvalues are presented as outputs, as we see below. As a comparison we also show the maximum eigenvalue obtained by using the `numpy`-provided function `numpy.linalg.eig`.



Power method show a relative error of 7.96×10^{-6}. However such seemingly large error is due to us setting the threshold to 10^{-6}. If we set a smaller threshold we see that the Power method actually produces a result much closer to the result of `numpy.linalg.eig`. The performance of the Lanczos method is much more interesting. The result coinsides with the true value up to 14 digits. Again, we cannot conclude hastily that the result of the Lanczos method agrees with that of the Power method, since we allowed the Power method to stop early with low accuracy. If we increase the accuracy of the Power method, we see an agreement.

It is also required to plot the eigenvalues of the computed tridiagonal matrix. The resulting plot is as the following figure.

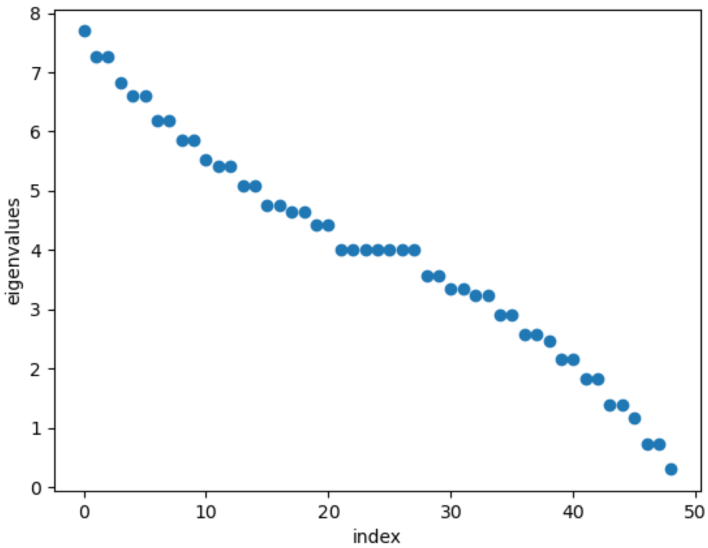


Figure 1: Scatter plot of the eigenvalues of A_7, in decreasing order

Appendix

The promised code, which computes the dimension of the Krylov space $K(e_1, A_7)$, is as follows.

```
import numpy as np
import sympy as sp

def make_Tn(n):
    if n <= 0 :
        raise KeyError

    off_main_diag = [-1 for _ in range(n-1)]
    upper = np.diag(off_main_diag, 1)
    lower = np.diag(off_main_diag, -1)
    return 4*np.eye(n) + upper + lower

def make_An(n):
    if n <= 0 :
        raise KeyError

    Tn = make_Tn(n)
    if n == 1 :
        return Tn

    I = np.eye(n)
    O = np.zeros((n,n))

    res = np.block([Tn, -I] + [O] * (n-2))
    for i in range(1, n-1):
        tmp = np.block([O] * (i-1) + [-I, Tn, -I] + [O] * (n-2-i))
        res = np.vstack((res, tmp))
    tmp = tmp = np.block([O] * (n-2) + [-I, Tn])
    res = np.vstack((res, tmp))
    return res
```

```
n = 7
A = sp.Matrix(np.array(make_An(n), dtype = int))
e1 = sp.Matrix(np.array(np.hstack(([1], np.zeros(n**2-1))), dtype = int))

K = e1
for i in range(1, 49):
    e1 = A * e1
    K = K.col_insert(i, e1)

print(K.rank())
```