NTNU

TMA4215 Numerical Mathematics — Fall 2016

Problem Set 8

1 a) Implement the power method given in eq. (5.17) in the textbook by Quarteroni *et al.* Test it on the matrix

$$A = \begin{bmatrix} -2 & -2 & 3 \\ -10 & -1 & 6 \\ 10 & -2 & -9 \end{bmatrix}$$

and use $\boldsymbol{x}^{(0)} = (1,0,0)$ as a starting value. Compare your result with that of MATLAB's built-in function `eig`.

Solution. One way to code the method is as follows, where `x` denotes the initial vector with Euclidean norm equal to 1 and `nmax` is the maximum number of iterations. We overwrite `x` as much as possible to save memory and also avoid 1 unnecessary computation of the product `A * x` with help of the temporary variable `z`.

```
function lambda_dominant = eig_power(A, x, nmax)
    z = A * x;
    for n = 1:nmax
        x = z;
        x = x / norm(x);
        z = A * x;
        lambda_dominant = dot(x, z);
    end
end
```

Matrix A has spectrum $\{-12, -3, 3\}$, which MATLAB's `eig` (almost) reproduces in double precision. 15 iterations of the power method yields the dominant eigenvalue $\lambda_{\text{dom}} \approx -12 + 9.313 \times 10^{-9}$.

Comment: Google's PageRank algorithm has—at least previously—employed the power method to rank websites in their search engine results.

b) When you are sure to have a working code, try it on the matrix

$$B = \begin{bmatrix} 5 & 1 & -1 \\ 1 & 11 & 7 \\ -1 & 7 & 11 \end{bmatrix}$$

using the same $\boldsymbol{x}^{(0)}$ as above. What is the result after 20 iterations? After 100? Explain what you observe.

Solution. Matrix B has eigenvalue spectrum $\{3, 6, 18\}$ and 20 iterations of the power method gives $\lambda_{\text{dom}} \approx 6 - 1.362 \times 10^{-12}$. Iterating 100 times, however, results in $\lambda_{\text{dom}} = 18$ in double precision. There are two reasons for this behavior: first, the power method assumes that $\alpha_{18} \neq 0$ in the representation

$$\boldsymbol{x}^{(0)} = \alpha_3 \boldsymbol{v}_3 + \alpha_6 \boldsymbol{v}_6 + \alpha_{18} \boldsymbol{v}_{18}$$

of the initial vector in terms of the eigenvectors $\{\boldsymbol{v}_i\}$ associated with the eigenvalues—indices correspond naturally to eigenvalues. But in this case,

$$\boldsymbol{v}_3 = (1, -1, 1), \qquad \boldsymbol{v}_6 = (2, 1, -1) \qquad \text{and} \qquad \boldsymbol{v}_{18} = (0, 1, 1),$$

so that $\alpha_3 = {}^1\!/_3 = \alpha_6$ and $\alpha_{18} = 0$. Hence, the power method—theoretically—actually finds the second dominant eigenvalue, which is 6; second, numerically, rounding errors gradually introduce a nonzero component along $\boldsymbol{v}_{18}$ in the sequence $\{\boldsymbol{x}^{(k)}\}$. This is picked up by the algorithm – think of it as restarting using a new $\boldsymbol{x}^{(0)}$ with a component in the direction of $\boldsymbol{v}_{18}$ – and gives convergence to the dominant eigenvalue 18.

c) Now implement the shifted inverse method—also called the inverse power method—in eq. (5.28) in Quarteroni *et al.* and test your function on the matrices from a) and b).

Solution. The inverse power method approximately finds the eigenvalue of a matrix A which is closest to a given number μ (`mu` in MATLAB) by applying the power method to the matrix $M_\mu^{-1} = (A - \mu I)^{-1}$. In order to save computations, we first compute the LU decomposition

$$P M_\mu = LU$$

of $M_\mu = A - \mu I$. The solution of $M_\mu \boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)}$ then becomes

$$\boldsymbol{x}^{(k)} = U^{-1}\left(L^{-1} P \boldsymbol{x}^{(k-1)}\right).$$

If μ already is an eigenvalue of A, the program terminates immediately. Moreover, as in a), `x` is overwritten as much as possible.

```
function lambda = eig_invpower(A, mu, x, nmax)
    [L, U, P] = lu(A - mu * eye(size(A)));
    assert(abs(prod(diag(U))) > 1e3 * eps, 'Stop: mu is an eigenvalue of A')
    for n = 1:nmax
        x = U \ (L \ (P * x));
        x = x / norm(x);
        lambda = dot(x, A * x);
    end
end
```

Several experiments on A and B indicate that the shifted inverse method seems to be efficient as long as μ is reasonably close to an exact eigenvalue. But, for example, with too few iterates and $\mu \gg 18$ for matrix B, the algorithm tends to find $\lambda \approx 6$ instead of $\lambda \approx 18$.

d) Apply the QR algorithm in eq. (5.32) in Quarteroni *et al.* on the two matrices from a) and b) and explain your findings.

Solution. A simple implementation of the QR algorithm is given in the listing to the left, while the right-hand side listing displays a more memory-efficient version. In both cases, the orthogonal input matrix Q is optional and defaults to the identity.

```
function T = eig_qr(A, nmax, Q)
    if (nargin == 3)
        T = Q' * A * Q;
    end
    for n = 1:nmax
       [Q, R] = qr(T);
       T = R * Q;
    end
end
```

```
function A = eig_qr(A, nmax, Q)
    if (nargin == 3)
        A = Q' * A * Q;
    end
    for n = 1:nmax
       [A, R] = qr(A);
       A = R * A;
    end
end
```

Testing on matrix B from b) gives

$$T \approx \begin{bmatrix} 6.0000 & 6.0416\times10^{-9} & -8.2863\times10^{-3} \\ 6.0416\times10^{-9} & 18.000 & 1.0938\times10^{-5} \\ -8.2863\times10^{-3} & 1.0938\times10^{-5} & 3.0000 \end{bmatrix}$$

to five significant figures after just 8 iterations, while matrix A yields

$$T \approx \begin{bmatrix} -12.000 & 10.733 & -7.6023 \\ -6.8243\times10^{-5} & -1.7999 & -1.6001 \\ 2.0472\times10^{-4} & -3.6002 & 1.8000 \end{bmatrix}.$$

Eigenvalues ±3 of A coincide—to four digits—with the eigenvalues of the lower right 2×2 submatrix of T.

2 Let I_n be the $n\times n$ identity matrix, $\boldsymbol{v}\in\mathbb{R}^n$ and $\theta\in\mathbb{R}$. Prove that the following matrices are orthogonal:

a) $\Omega = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$

b) $Q = I_n - 2\frac{1}{\boldsymbol{v}^\top\boldsymbol{v}}\boldsymbol{v}\boldsymbol{v}^\top.$

Also, show that Q is symmetric.

Solution. a) Using the identity $\cos^2\theta+\sin^2\theta=1$, it is immediate that $\Omega^\top\Omega = I_2$.

b) Since I_n is symmetric and $\left(\boldsymbol{v}\boldsymbol{v}^\top\right)^\top = \boldsymbol{v}^{\top\top}\boldsymbol{v}^\top = \boldsymbol{v}\boldsymbol{v}^\top$, it follows that Q is symmetric. Hence, utilizing

$$\left(\boldsymbol{v}\boldsymbol{v}^\top\right)\left(\boldsymbol{v}\boldsymbol{v}^\top\right) = \boldsymbol{v}\left(\boldsymbol{v}^\top\boldsymbol{v}\right)\boldsymbol{v}^\top = \left(\boldsymbol{v}^\top\boldsymbol{v}\right)\boldsymbol{v}\boldsymbol{v}^\top,$$

because $\boldsymbol{v}^\top\boldsymbol{v}$ is a scalar, yields

$$Q^\top Q = Q^2 = I_n - 4\frac{1}{\boldsymbol{v}^\top\boldsymbol{v}}\boldsymbol{v}\boldsymbol{v}^\top + 4\frac{1}{(\boldsymbol{v}^\top\boldsymbol{v})^2}\left(\boldsymbol{v}\boldsymbol{v}^\top\right)\left(\boldsymbol{v}\boldsymbol{v}^\top\right) = I_n.$$

3 Find by hand the reduced QR factorization of the matrix

$$A = \begin{bmatrix} 4 & 4 \\ 0 & 2 \\ 3 & 3 \end{bmatrix}$$

using the Gram–Schmidt orthogonalization process.

Solution. Let $\boldsymbol{a}_1 = (4,0,3)$ and $\boldsymbol{a}_2 = (4,2,3)$ be the two columns of A. Then the Gram–Schmidt process works as follows:

$$\boldsymbol{u}_1 = \boldsymbol{a}_1, \qquad \boldsymbol{q}_1 = \frac{\boldsymbol{u}_1}{\|\boldsymbol{u}_1\|_2},$$

$$\boldsymbol{u}_2 = \boldsymbol{a}_2 - \langle \boldsymbol{a}_2, \boldsymbol{q}_1 \rangle \boldsymbol{q}_1, \qquad \boldsymbol{q}_2 = \frac{\boldsymbol{u}_2}{\|\boldsymbol{u}_2\|_2},$$

and $\{\boldsymbol{q}_1, \boldsymbol{q}_2\}$ is an orthonormal basis for the column space of A. Moreover, we can express $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ as

$$\boldsymbol{a}_1 = \langle \boldsymbol{a}_1, \boldsymbol{q}_1 \rangle \boldsymbol{q}_1 \qquad \text{and} \qquad \boldsymbol{a}_2 = \langle \boldsymbol{a}_2, \boldsymbol{q}_1 \rangle \boldsymbol{q}_1 + \langle \boldsymbol{a}_2, \boldsymbol{q}_2 \rangle \boldsymbol{q}_2.$$

This can be written in matrix form as $A = QR$, where

$$Q = [\boldsymbol{q}_1 \; \boldsymbol{q}_2] \qquad \text{and} \qquad R = \begin{bmatrix} \langle \boldsymbol{a}_1, \boldsymbol{q}_1 \rangle & \langle \boldsymbol{a}_2, \boldsymbol{q}_1 \rangle \\ 0 & \langle \boldsymbol{a}_2, \boldsymbol{q}_2 \rangle \end{bmatrix}.$$

Routine calculations now yield

$$Q = \begin{bmatrix} 4/5 & 0 \\ 0 & 1 \\ 3/5 & 0 \end{bmatrix} \qquad \text{and} \qquad R = \begin{bmatrix} 5 & 5 \\ 0 & 2 \end{bmatrix}.$$

4 The Householder QR factorization of $A \in \mathbb{R}^{n\times m}$, where $n \geq m$, is given by the following algorithm:

for $k = 1$ to m **do**

$\quad \boldsymbol{x} \leftarrow A_{k:n,k}$

$\quad \boldsymbol{v}_k := \boldsymbol{x} + \operatorname{sign}(x_1)\|x\|_2 \boldsymbol{e}_1$

$\quad \boldsymbol{v}_k \leftarrow \boldsymbol{v}_k / \|\boldsymbol{v}_k\|_2$

$\quad A_{k:n,k:m} \leftarrow A_{k:n,k:m} - 2\boldsymbol{v}_k \left(\boldsymbol{v}_k^\top A_{k:n,k:m}\right)$

end for

Here $A_{i:j,k:\ell}$—omitting colon(s) if $i = j$ and/or $k = \ell$—denotes the elements of A in accordance with MATLAB's indexing convention. This algorithm transforms A into the upper trapezoidal matrix $R \in \mathbb{R}^{n\times m}$ in the QR decomposition, and is implemented in the attached file `householder.m`. It does not, however, produce Q, but we know that

$$Q = Q_1 Q_2 \cdots Q_m,$$

where

$$Q_1 = I_n - 2\boldsymbol{v}_1\boldsymbol{v}_1^\top \qquad \text{and} \qquad Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & I_{n-k+1} - 2\boldsymbol{v}_k\boldsymbol{v}_k^\top \end{bmatrix} \quad \text{for} \quad k = 2, \ldots, m.$$

a) Apply the algorithm first by hand and then with `householder.m` to the matrix in Exercise 3. Also, find Q_1 and Q_2, and compute Q. Compare with the results from MATLAB's built-in function `qr`.

Solution. Hand calculations and `householder.m` both give

$$x = (4,0,3), \qquad v_1 = (9,0,3), \qquad v_1 \leftarrow (3,0,1)/\sqrt{10}, \qquad A \leftarrow \begin{bmatrix} -5 & -5 \\ 0 & 2 \\ 0 & 0 \end{bmatrix},$$

$$x = (2,0), \qquad v_2 = (4,0), \qquad v_2 \leftarrow (1,0), \qquad R = A \leftarrow \begin{bmatrix} -5 & -5 \\ 0 & -2 \\ 0 & 0 \end{bmatrix}.$$

Moreover, we find that

$$Q_1 = \begin{bmatrix} -4/5 & 0 & -3/5 \\ 0 & 1 & 0 \\ -3/5 & 0 & 4/5 \end{bmatrix}, \qquad Q_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad Q = Q_1Q_2 = \begin{bmatrix} -4/5 & 0 & -3/5 \\ 0 & -1 & 0 \\ -3/5 & 0 & 4/5 \end{bmatrix},$$

and both Q and R match exactly the output from `qr`, except for an unimportant change of signs in the second column of Q and $R_{2,2}$.

b) Let $\boldsymbol{x} \in \mathbb{R}^n$ and remember that $Q \in \mathbb{R}^{n\times n}$. Show that the product $Q\boldsymbol{x}$ is performed by the following procedure:

for $k = m$ downto 1 **do**
  $\boldsymbol{x}_{k:n} \leftarrow \boldsymbol{x}_{k:n} - 2\boldsymbol{v}_k(\boldsymbol{v}_k^\top \boldsymbol{x}_{k:n})$
end for

How can we use this to form Q itself?

Solution. Since $Q = Q_1 \cdots Q_m$, the product $Q\boldsymbol{x}$ can be calculated recursively as

$$\boldsymbol{x} \leftarrow Q_k \boldsymbol{x} \qquad \text{for} \qquad k = m, \ldots, 1,$$

where we overwrite $\boldsymbol{x}$ in each step. By definition, Q_k leaves the first $k-1$ components of $\boldsymbol{x}$ unchanged, whereas the latter components become

$$\boldsymbol{x}_{k:n} \leftarrow \left(I_{n-k+1} - 2\boldsymbol{v}_k\boldsymbol{v}_k^\top\right)\boldsymbol{x}_{k:n} = \boldsymbol{x}_{k:n} - 2\boldsymbol{v}_k(\boldsymbol{v}_k^\top \boldsymbol{x}_{k:n}).$$

In total, this gives the procedure above. As regards how to form Q itself, observe first that

$$Q = QI_n = Q\begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \ldots & \boldsymbol{e}_n \end{bmatrix} = Q\boldsymbol{e}_1 + Q\boldsymbol{e}_2 + \cdots + Q\boldsymbol{e}_n,$$

where $\boldsymbol{e}_i$ is the ith column of I_n. Therefore, we can apply the given algorithm to the $\boldsymbol{e}_i$'s to get all the columns $Q\boldsymbol{e}_i$ of Q.

c) Extend `householder.m` to also return Q, and compare your result with `qr`.

Solution. Using the idea in b), we can extend `householder.m` as follows.

```
function [Q, R] = householder(A)
    % Householder QR decomposition.
    [n, m] = size(A);

    % Compute R and store the v_k vectors in V.
    V = zeros(n, m);
    for k = 1:m
        x = A(k:n, k);
        v = x;
        v(1) = x(1) + sign(x(1)) * norm(x);
        v = v / norm(v);
        V(k:n, k) = v;
        A(k:n, k:m) = A(k:n, k:m) - 2 * v * (v' * A(k:n, k:m));
    end
    R = A;

    % Compute Q.
    Q = eye(n);
    for i = 1:n
        for k = m:-1:1
            Q(k:n, i) = Q(k:n, i) - 2 * (V(k:n, k)' * Q(k:n, i)) * V(k:n, k);
        end
    end
end
```

The output when applied to the matrix in Exercise 3 is identical to that of `qr`, except for the insignificant change of signs observed in a).

d) Let A be the Hilbert matrix of dimension 8 – you can build it in MATLAB by typing `A = hilb(8)`. Find the QR decomposition of A both via the Gram–Schmidt orthogonalization process—included in the attached file `GramSchmidt.m` for convenience—and the Householder transformations. How well will $Q^\top Q$ approximate I_8 in the two cases?

Solution. We omit printing the numerical values for Q and R, but note that R from the Gram–Schmidt process by construction at least has perfect zeros on its subdiagonal. One way to measure how well $Q^\top Q$ approximates I_8 is to compute $\|Q^\top Q - I_8\|$ in some norm. In the 2-norm, for example, we find that

$$\text{Householder:} \quad \|Q^\top Q - I_8\|_2 \approx 2.3535 \times 10^{-15},$$

$$\text{and Gram–Schmidt:} \quad \|Q^\top Q - I_8\|_2 \approx 1.0323.$$

As seen, Householder transformations are superior to the basic Gram–Schmidt algorithm, which is inherently numerically unstable.