# Lab Class (Eigenvalue Problems)

## 1 Review of methods to compute eigenvalues and eigenvectors

**Power method** The power method is the iteration

$$x_{k+1} = Ax_k, \quad k = 0, 1, 2, \ldots,$$

starting from a given vector $x_0$. To prevent overflow / underflow, we rescale the resulting vector after each iteration so that it has norm 1; i.e., we modify the iteration to

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|}, \quad k = 0, 1, 2, \ldots.$$

The power method converges (under suitable assumptions, and in a suitable sense to keep track of normalization) to the eigenvector $v_1$ relative to the eigenvalue $\lambda_1$ of $A$ with maximum modulus.

**Inverse iteration** Inverse iteration means applying the power method to $B = (A - tI)^{-1}$, for a given $t \in \mathbb{C}$. The eigenvalues of $B$ are given by $\frac{1}{\lambda_i - t}$, where $\lambda_i$ are the eigenvalues of $A$. Hence its eigenvalue of maximum modulus corresponds to the eigenvalue $\lambda_i$ which is closest to $t$.

**Orthogonal (subspace) iteration** Starting from a given $\hat{Z}_0 \in \mathbb{C}^{n \times p}$, with $n \geq p$, orthogonal iteration corresponds to performing at each step a thin QR factorization

$$\underbrace{\hat{Z}_{k+1}\hat{R}_{k+1}}_{\text{thin QR}} = A\hat{Z}_k, \quad , k = 0, 1, 2, \ldots,$$

where for each $k$ the matrix $\hat{Z}_k \in \mathbb{C}^{n \times p}$ has orthonormal columns, and $\hat{R}_k \in \mathbb{C}^{p \times p}$ is upper triangular. Under suitable assumptions, $\hat{Z}_k$ converges (in a suitable sense) to a basis of the invariant subspace $\text{span}(v_1, v_2, \ldots, v_p)$, where $v_1, \ldots, v_p$ are the eigenvectors of $A$ corresponding to its $p$ largest (in absolute value) eigenvalues.

**Orthogonal iteration with $p = n$** When one runs orthogonal iteration with $p = n$, starting from $\hat{Z}_0 = I$, the $n \times n$ matrix $\hat{Z}_k^* A \hat{Z}_k$ converges (under suitable assumptions) to a triangular matrix.

**QR iteration** Shifted QR iteration is the iteration $[Q_k, R_k] = \texttt{qr}(A_{k-1} - \sigma_k I)$, $A_k = R_k Q_k + \sigma_k I$. At each step, we will choose $\sigma_k$ as the bottom right entry $A_{k-1}(end, end)$ of $A_{k-1}$[1].

## 2 Power method

We shall use the following function to generate random matrices with prescribed eigenvalues.

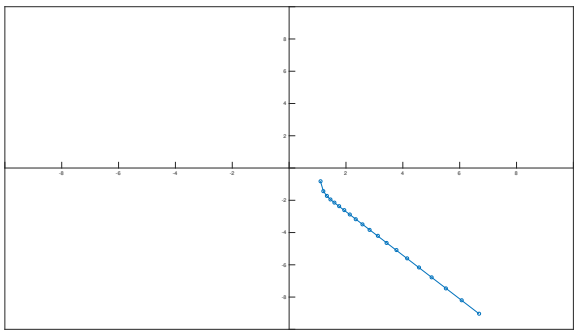
```
function A = random_matrix_with_eigenvalues(ev)
% given a vector ev, returns a matrix whose eigenvalues are its entries.
V = randn(length(ev));
A = V * diag(ev) / V;
```

You can check in a few examples that the eigenvalues of $A$ coincide with the elements of `ev`.

Moreover, before starting with the exercises, type at the Matlab prompt the command `format short e`. This instruction tells Matlab to use exponential notation to display each matrix entry separately. It makes it easier to assess the magnitude of each entry.

1. We want to plot (as points in the plane $\mathbb{R}^2$) the first iterates of the power method (without normalization) for a $2 \times 2$ matrix. Generate a matrix `A = random_matrix_with_eigenvalues([1.1, 0.3])` and a random vector $z_0 \in \mathbb{R}^2$. Then, compute the vectors $z_0, Az_0, A^2 z_0, \ldots, A^{20} z_0 \in \mathbb{R}^2$. Save their first entry in $x \in \mathbb{R}^{21}$, and their second entry in $y \in \mathbb{R}^{21}$, and then display them with `plot(x, y, 'o-');`.

   You should see that the vectors tend to line up along the same direction, which is the one of the eigenvector $v_1$ of $A$. An example is in the following picture.



[1] Note that this strategy will not lead to convergence if $A$ has complex eigenvalues, because all the entries that we compute stay in $\mathbb{R}$. In this case, we need a different shifting strategy, such as taking $\sigma_k$ as one of the eigenvalues of the $2\times 2$ matrix $A_{k-1}(end-1 : end, end-1 : end)$.

2. Write a function `function [v, lambda] = powermethod(A)` that performs 100 steps of the power method, with normalization, starting from a random vector $x_0$. At the end of the function, right before it ends, plot the value of the residual $\|Ax_k - (\frac{x_k^* A x_k}{x_k^* x_k})x_k\|$ as a function of the step number $k$: after generating a $100 \times 1$ array `residuals`, Use the instruction `semilogy(1:100, residuals)`, which produces a plot using a logarithmic scale on the $y$ axis.

3. Test the previous function on matrices with different ratios $\lambda_2/\lambda_1$; for instance, `random_matrix_with_eigenvalues(ev)` with `ev=[2,1,0.5,0]`, `ev=[-10,1,0.2]`, or `[2,1.9,-0.5,0]`, or `[2,-1.9,0.5,0]`. You should see from the plot that the convergence speed is faster for the first two cases and slower in the other two (indeed, it depends on the ratio $\frac{\lambda_2}{\lambda_1}$).

## 3 Inverse iteration

1. Write a function `function [v, lambda] = inverse_iteration(A, t)` that executes 10 steps of inverse iteration (starting from a random initial value $x_0$), and returns the corresponding estimate of the closest eigenvalue $\lambda$ of $A$ to $v$. Compute the LU factorization $A = LU$ only once with `[L, U] = lu(A - t*eye(length(A)))` at the beginning of the algorithm, and then at each step use the factors $L, U$ to solve the linear systems.

2. Test your function on `A = random_matrix_with_eigenvalues(-5:5)`, and choose the shift to get convergence to an internal eigenvalue, for instance $t = 1.7$. How accurate is the computed eigenvalue?

## 4 Orthogonal iteration

We shall write code for the orthogonal iteration only for the case $p = n$, for simplicity. Recall that the Matlab command to compute the thin QR factorization of a matrix $M$ is `[Zhat, Rhat] = qr(M, 0)`.

1. Write a function `function orthogonal_iteration(A, kmax)` that performs $k_{\max}$ iterations of orthogonal iteration, with $p = n$, starting from $\hat{Z}_0 = I$, and after each step prints the matrix $A_k = \hat{Z}_k^* A \hat{Z}_k$.

2. Test the function on `A = random_matrix_with_eigenvalues(-4.7:4.3)`, to verify that the method indeed converges to the Schur form of $A$.

3. Test the method on `A = random_matrix_with_eigenvalues(-5:5)`. Note that this matrix has several pairs of eigenvalues with the same absolute value. Which entries converge to zero and which do not?

## 5 Shifted QR

1. Write a function `function qr_iteration(A, kmax)` that performs $k_{\max}$ steps of QR iteration on the matrix $A$. Take as shift the last entry $A_{k-1}(end, end)$ of $A_{k-1}$. Again, after each step print the matrix $A_k$.

2. Test the function on `A = random_matrix_with_eigenvalues(-5:5)`. You should see that the entries in the last row of $A_k$ (apart from the last one) decrease sharply. For instance, in one of the random-generated examples I have obtained

```
>> qr_iteration(A, 10)
A_1(end,1) = 0.290177
A_2(end,1) = -0.0481675
A_3(end,1) = 0.00270503
A_4(end,1) = -2.39121e-06
A_5(end,1) = 1.32516e-11
A_6(end,1) = -1.09329e-20
A_7(end,1) = 8.60834e-38
A_8(end,1) = -5.31815e-71
A_9(end,1) = 1.98253e-136
A_10(end,1) = -2.70929e-266
```

   The exponents in the sequence $-6, -11, -20, -38, -71, -136, -266$ approximately double at each step. This kind of convergence (called *quadratic convergence*) is much faster than what we have seen in the previous algorithms.

3. Implement *deflation* in your QR iteration: once the convergence criterion `norm(Ak(end, 1:end-1)) <= eps * norm(A)` is satisfied, you have found a matrix $A_k$ which has the same eigenvalues as $A$ and is essentially in the form

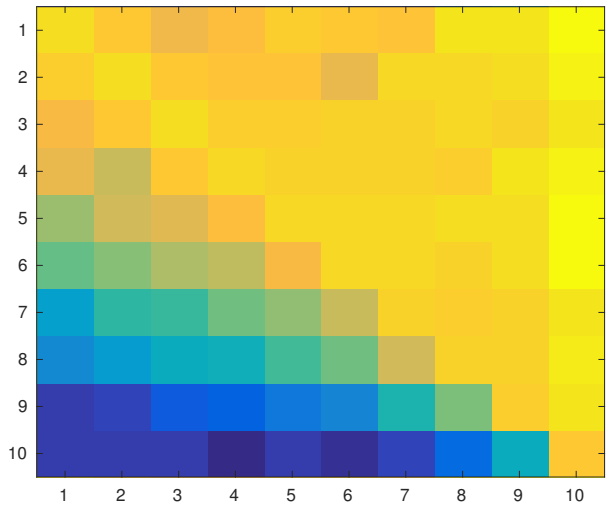
$$A_k = \begin{bmatrix} B & c \\ 0 & d \end{bmatrix},$$

   with $B \in \mathbb{C}^{(n-1)\times(n-1)}$ and $d \in \mathbb{C}^{1\times 1}$. Hence the eigenvalues of $A$ are given by the union of the eigenvalues of $B$ and the complex number $d$. Hence we can replace replace $A_k$ with $B$ and continue on this submatrix. Keep track of the eigenvalues $d$ that you have computed at each step and return them all together at the end of the function. This procedure should return the exact eigenvalues of $A$.

## 6 Additional exercises

1. Verify on an example that orthogonal iteration and QR iteration (with shifts $\sigma_k = 0$) give the same matrices $A_k$.

2. In `orthogonal_iteration` and `qr_iteration`, instead of printing the matrices `Ak` at each step, display them visually using the command `imagesc(log(abs(Ak)))`. This command uses squares in different colors to display the absolute value of each element of $A$ (in a logarithmic scale). An example is in the following picture.




   To be able to see clearly the pictures after each iteration, you may want to introduce a short pause with the command `pause(0.2)`.

3. Modify `qr_iteration` so that you use as shift the smallest eigenvalue of the $2\times 2$ matrix `S = Ak(end-1:end, end-1:end)`, as suggested in the first page. (In this exercise you can be lazy and use Matlab's function `eig()` on S, but in order to get a self-contained function to find eigenvalues in theory one should compute them by solving the quadratic equation $\det(S-xI) = 0$.) Test the resulting method on a matrix having non-real eigenvalues; for instance, generate them randomly with `A = randn(10, 10)`. A random matrix generated in this way has nonreal eigenvalues the vast majority of the times. Compare the eigenvalues that you have computed with the output of Matlab's own `eig(A)`.

4. Modify `qr_iteration` so that instead of removing the last row after it has converged, you keep the whole matrix in memory but you find at each step the QR factorization of its top "active block" `B = Ak(1:p, 1:p)`. Note that the proper way of applying the $p \times p$ factor $Q_k$ that you have obtained is using the similarity transformation

$$\begin{bmatrix} Q_k^* & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B & C \\ 0 & D \end{bmatrix} \begin{bmatrix} Q_k & 0 \\ 0 & I \end{bmatrix},$$

which modifies the block labeled $C$ as well. If implemented properly, this procedure should end by computing an upper triangular matrix which is similar to $A$, i.e., it should compute its Schur factorization.