Problem sheet 1

Joe & Víctor

Department of Aeronautics
Imperial College London

The aim of these problems is to familiarize yourselves with the mathematical tools that will be used throughout the project. To check the solution we encourage you to solve the problems using `Python`. Hopefully we will provide with everything you need to get started. Ideally you should try to do this problems by your own and of course if you have doubts talk between yourselves or write to us. Later on in the project we might explore the possibility of doing divide and conquer, but for the moment we think that it is better that you solidify your knowledge in these basic concepts.

We talked with Joe after last meeting and we both think that it's better for you to try to solve these small little problems that will be useful for the set up of the input for our computations, rather than try to fully understand how those methods work. We are open that you explore and code by yourselves any of the methods that are already in the literature so that later we have more data to compare with, but in general we think that for most of the methods an intuitive idea is enough (it is what we always have in mind!! We don't remember the details of LU decomposition for example).

Elemental matrix operations

Exercise 1. Consider the matrices

$$\mathbf{A} = \begin{pmatrix} 7 & 2 \\ 3 & 6 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 2 & -2 \\ 1 & 3 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & -6 & 7 \\ 1 & 8 & 9 \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} 4 & 5 & 6 \\ 3 & -1 & 2 \\ 1 & 6 & 4 \end{pmatrix}.$$

1. Compute $\mathbf{A} + \mathbf{B}$, $\mathbf{AB}$, $\mathbf{CD}$.

   Note. In `Python`, matrices can be defined as an array using the `numpy` library. For example, the matrix $\mathbf{A}$ can be defined as

   ```
   import numpy as np
   A = np.array([[7, 2],[3, 6]])
   ```

   Matrix addition can be performed using the `+` operator, while matrix multiplication can be performed using the `@` operator. The operator `*` performs element-wise multiplication, e.g. `A * A` will result in the matrix

   $$\begin{pmatrix} 49 & 4 \\ 9 & 36 \end{pmatrix},$$

   while `A @ A` will result in

   $$\begin{pmatrix} 55 & 26 \\ 39 & 42 \end{pmatrix}.$$

2. Remember that a necessary and sufficient condition for a matrix to be invertible is that its determinant is non-zero. Compute the determinants of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ and check which of these matrices are invertible. For those that are, compute their inverses.

   Note. Let's start using the `scipy` library to compute determinants and inverses. For this we need to import the `linalg` module from `scipy` as follows:

```
import scipy.linalg as la
```

Then, the determinant of a matrix `M` can be computed using

```
det_M = la.det(M)
```

and if the matrix is invertible, its inverse can be computed using

```
inv_M = la.inv(M)
```

3. A pair $(\lambda, \mathbf{v})$, where λ is a scalar (e.g. $\lambda \in \mathbb{R}$) and $\mathbf{v}$ is a vector (e.g. $\mathbf{v} \in \mathbb{R}^n$), is called an eigenvalue-eigenvector pair of a square matrix $\mathbf{M}$ if it satisfies the equation

$$\mathbf{M}\mathbf{v} = \lambda\mathbf{v}.$$

It can be seen that a necessary condition for λ to be an eigenvalue of $\mathbf{M}$ is that $p(\lambda) = \det(\mathbf{M} - \lambda\mathbf{I}) = 0$, where $\mathbf{I}$ is the identity matrix of the same size as $\mathbf{M}$. Note that $p(\lambda)$ is a polynomial of degree n if $\mathbf{M} \in \mathbb{R}^{n\times n}$, called the *characteristic polynomial* of $\mathbf{M}$.

Recall that even if a matrix has real entries, its eigenvalues and eigenvectors may be complex. In this case, we will also have their conjugates as eigenvalue-eigenvector pairs (remember if $z = a + bi \in \mathbb{C}$, its conjugate is $\bar{z} = a - bi$). This is because if $\mathbf{M}\mathbf{v} = \lambda\mathbf{v}$, then taking conjugates on both sides gives $\overline{\mathbf{M}\mathbf{v}} = \overline{\lambda\mathbf{v}}$, which implies that $\mathbf{M}\overline{\mathbf{v}} = \overline{\lambda}\overline{\mathbf{v}}$ since $\mathbf{M}$ has real entries (e.g. $\overline{\mathbf{M}} = \mathbf{M}$).

Compute the characteristic polynomial of all the matrices above and find the eigenvalues by finding the roots of these polynomials.

Note. In `Python`, the eigenvalues and eigenvectors of a matrix `M` can be computed using the function `eig` from the `linalg` module as follows:

```
eigenvalues, eigenvectors = la.eig(M)
```

The variable `eigenvalues` will contain the eigenvalues of the matrix, while the variable `eigenvectors` will contain the corresponding eigenvectors as columns.

Note that `Python` uses the letter `j` to represent the imaginary unit i. E.g. to introduce the complex number $3 + 4\mathrm{i}$, you would write

```
z = 3 + 4j
```

4. In the case of having n distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ with corresponding eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ of a matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, it can be shown that if we define the matrix

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \ldots & \mathbf{v}_n \\ | & | & & | \end{pmatrix}$$

(i.e. the matrix whose columns are the eigenvectors of $\mathbf{M}$) and the diagonal matrix

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix},$$

then the following relation holds:

$$\mathbf{M} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}.$$

This is called the *eigendecomposition* of $\mathbf{M}$.

For those matrices with n distinct eigenvalues, compute their eigendecomposition and verify that the relation above holds. (Be careful with the ordering of the eigenvalues and eigenvectors in the matrices $\mathbf{\Lambda}$ and $\mathbf{V}$! The order doesn't matter as long as the pairs are kept together, e.g. if λ_i is the j-th eigenvalue, then $\mathbf{v}_i$ must be the j-th eigenvector).

Note. Note that the matrix `eigenvectors` obtained from the function `eig` already corresponds to the matrix $\mathbf{V}$ defined above. The matrix $\mathbf{\Lambda}$ can be constructed using the function `diag` from the `numpy` library as follows:

```
Lambda = np.diag(eigenvalues)
```

5. An important property of determinants is that if $\mathbf{M}_1, \mathbf{M}_2$ are square matrices of the same size, then

$$\det(\mathbf{M}_1\mathbf{M}_2) = \det(\mathbf{M}_1)\det(\mathbf{M}_2).$$

From here, it follows that

$$1 = \det(\mathbf{I}) = \det\left(\mathbf{M}\mathbf{M}^{-1}\right) = \det(\mathbf{M})\det\left(\mathbf{M}^{-1}\right),$$

which implies that

$$\det\left(\mathbf{M}^{-1}\right) = \frac{1}{\det(\mathbf{M})}.$$

We can apply this property to the eigendecomposition of a matrix $\mathbf{M}$ as follows:

$$\det(\mathbf{M}) = \det\left(\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}\right) = \det(\mathbf{V})\det(\mathbf{\Lambda})\det\left(\mathbf{V}^{-1}\right) = \det(\mathbf{V})\det(\mathbf{\Lambda})\det(\mathbf{V})^{-1} = \det(\mathbf{\Lambda}) = \lambda_1\lambda_2\cdots\lambda_n,$$

where in the last equality we have used that the determinant of a diagonal matrix is the product of its diagonal entries.

Using the property above, compute the determinants of the matrices using their eigenvalues and verify that they match the values computed in part (2).

Note. In `Python`, and particularly in `numpy`, we want to avoid for loops as much as possible for efficiency reasons. To compute the sum or product of an array `A`, you can use the function `sum` or `prod`, respectively, from the `numpy` library as follows:

```
sum_A = np.sum(A)
prod_A = np.prod(A)
```

6. **Important!** Can you think in a way of generating invertible random matrices of size n with eigenvalues in the interval (a, b)? (this will be useful later on when generating test cases). We can set $n = 3$ (for the moment) $a = 0.1$ and $b = 5$. For the sake of numerical stability we don't want eigenvalues too close to zero neither too large.

Note. Hint: you can generate random numbers in `Python` using the `random` module from the `numpy` library. For example, to generate a random number between 0 and 1, you can use

```
num = np.random.rand()
```

and to generate a random number between a and b, you can use

```
num = a + (b - a) * np.random.rand()
```

7. **Important!** What if we want a random number of eigenvalues $0 \leq m \leq n$ (with n the size of the matrix) to be in the interval (a, b) and the rest, e.g. $n - m$, to be in the interval (c, d)?

Norms and orthogonality

Exercise 2. Consider the vectors

$$\mathbf{u} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} -2 \\ 1 \\ 8 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}.$$

1. We define the *inner product* (or *dot product*) of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ as the matrix multiplication

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i,$$

Here a_i and b_i denote the i-th components of the vectors $\mathbf{a}$ and $\mathbf{b}$, respectively. We say that two vectors are *orthogonal* if their inner product is zero, i.e. $\langle \mathbf{a}, \mathbf{b} \rangle = 0$.

We define the *norm* of a vector $\mathbf{a} \in \mathbb{R}^n$ as

$$\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle} = \sqrt{\sum_{i=1}^{n} a_i^2}.$$

Compute $\langle \mathbf{u}, \mathbf{v} \rangle$, $\langle \mathbf{u}, \mathbf{w} \rangle$ and $\langle \mathbf{v}, \mathbf{w} \rangle$. Are any of these vectors orthogonal to each other? Compute the norms $\|\mathbf{u}\|$, $\|\mathbf{v}\|$ and $\|\mathbf{w}\|$.

Note. In `Python`, the inner product of two vectors `a` and `b` can be computed using the function `dot` from the `numpy` library as follows:

```
inner_product = np.dot(a, b)
```

The norm of a vector `a` can be computed using the function `norm` from the `linalg` module of the `scipy` library as follows:

```
norm_a = la.norm(a)
```

2. We say that a vector $\mathbf{a}$ is linearly dependent on a set of vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m\}$ if there exist scalars $c_1, c_2, \dots, c_m$ such that

$$\mathbf{a} = c_1\mathbf{b}_1 + c_2\mathbf{b}_2 + \cdots + c_m\mathbf{b}_m.$$

If no such scalars exist, then we say that $\mathbf{a}$ is linearly independent of the set $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m\}$. Since the determinant of a matrix is invariant under column operations, it can be shown that a set of n vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$ in $\mathbb{R}^n$ is linearly independent if and only if the matrix

$$\mathbf{B} = \begin{pmatrix} | & | & & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \dots & \mathbf{b}_n \\ | & | & & | \end{pmatrix}$$

is invertible (i.e. $\det(\mathbf{B}) \neq 0$).

Arrange these vectors as columns of a matrix

$$\mathbf{M} = \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{pmatrix}$$

and compute the matrix $\det(\mathbf{M})$. Compute the determinants of the following matrices as well:

$$\mathbf{M}_1 = \begin{pmatrix} | & | & | \\ \mathbf{u}+4\mathbf{v} & \mathbf{v} & \mathbf{w}-\mathbf{u} \\ | & | & | \end{pmatrix}, \quad \mathbf{M}_2 = \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v}-3\mathbf{u} & \mathbf{w}+2\mathbf{v}+7\mathbf{u} \\ | & | & | \end{pmatrix}.$$

What do you observe?

Note. If you have defined the vectors `u`, `v` and `w` as row vectors (e.g. `u = np.array([1, 2, -1])`), you can arrange them as columns of a matrix `M` as follows:

```
M = np.column_stack((u, v, w))
```

3. Gram-Schmidt orthogonalization is a method for generating an orthogonal set of vectors from a linearly independent set of vectors. Given a set of linearly independent vectors $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m\}$, we can generate an orthogonal set of vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\}$ as follows:

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{b}_1, \\
\mathbf{u}_2 &= \mathbf{b}_2 - \frac{\langle \mathbf{u}_1, \mathbf{b}_2 \rangle}{\langle \mathbf{u}_1, \mathbf{u}_1 \rangle}\mathbf{u}_1, \\
\mathbf{u}_3 &= \mathbf{b}_3 - \frac{\langle \mathbf{u}_1, \mathbf{b}_3 \rangle}{\langle \mathbf{u}_1, \mathbf{u}_1 \rangle}\mathbf{u}_1 - \frac{\langle \mathbf{u}_2, \mathbf{b}_3 \rangle}{\langle \mathbf{u}_2, \mathbf{u}_2 \rangle}\mathbf{u}_2, \\
&\vdots \\
\mathbf{u}_m &= \mathbf{b}_m - \sum_{j=1}^{m-1} \frac{\langle \mathbf{u}_j, \mathbf{b}_m \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle}\mathbf{u}_j.
\end{aligned}$$

In orther for them to be an orthonormal set (which means orthogonal and unit norm for each vector), we can normalize each vector as well:

$$\mathbf{e}_i = \frac{\mathbf{u}_i}{\|\mathbf{u}_i\|}, \quad i = 1, 2, \ldots, m.$$

So that $\langle \mathbf{e}_i, \mathbf{e}_j \rangle = 0$ for $i \neq j$ and $\|\mathbf{e}_i\| = 1$ for all i.

Check theoretically that for all $i, j = 1, 2, \ldots, m$ with $i \neq j$, we have that $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = 0$. Hint: Apply the formulas above and use the properties of the inner product (check wikipedia if you need to)

4. Apply the Gram-Schmidt process to the set of vectors $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ defined above to generate an orthonormal set of vectors $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$. Verify that these vectors are indeed orthogonal to each other by computing their inner products. You should get $\langle \mathbf{a}, \mathbf{b} \rangle = \langle \mathbf{a}, \mathbf{c} \rangle = \langle \mathbf{b}, \mathbf{c} \rangle = 0$. Check as well that their norms are unitary, e.g. $\|\mathbf{a}\| = \|\mathbf{b}\| = \|\mathbf{c}\| = 1$.

5. Given a matrix

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{pmatrix},$$

we define its transpose as

$$\mathbf{M}^{\mathrm{T}} = \begin{pmatrix} m_{11} & m_{21} & \cdots & m_{n1} \\ m_{12} & m_{22} & \cdots & m_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{1n} & m_{2n} & \cdots & m_{nn} \end{pmatrix}.$$

(we send rows to columns and vice versa, or equivalently we mirror along the main diagonal). We say that a matrix is *symmetric* if $\mathbf{M} = \mathbf{M}^{\mathrm{T}}$.

Compute the transpose of the matrix

$$\mathbf{N} = \begin{pmatrix} | & | & | \\ \mathbf{a} & \mathbf{b} & \mathbf{c} \\ | & | & | \end{pmatrix},$$

where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are the orthonormal vectors obtained in the previous part.

Note. In `Python`, the transpose of a matrix `M` can be computed using the attribute `T` as follows:

```
M_transpose = M.T
```

6. Compute the inverse of $\mathbf{N}$. Does it sound familiar the expression you get? Solution: you should get that $\mathbf{N}^{-1} = \mathbf{N}^{\mathrm{T}}$.

7. Let's think now that $\mathbf{N}$ does the role of $\mathbf{V}$ in the eigendecomposition of a matrix $\mathbf{M}$ in the previous exercise, e.g.

$$\mathbf{M} = \mathbf{N}\mathbf{\Lambda}\mathbf{N}^{-1} = \mathbf{N}\mathbf{\Lambda}\mathbf{N}^{\mathrm{T}},$$

   where $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues of $\mathbf{M}$ in its diagonal.

   Prove that in this case, the matrix $\mathbf{M}$ is symmetric, e.g. $\mathbf{M} = \mathbf{M}^{\mathrm{T}}$. Hint: use the property of the transpose operation that states that $(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$. First try to do it mathematically formal and general, then check it with the matrix $\mathbf{N}$ computed before and any diagonal matrix $\mathbf{\Lambda}$ of your choice.

8. **Important!** How would you adapt the method in the 6th section of exercice 1 to generate random symmetric matrices with eigenvalues in the interval (a, b)?