11. Matrix inverses

11.1. Left and right inverses

We'll see later how to find a left or right inverse, when one exists.

In [ ]:
```
A = np.array([[-3,-4], [4,6], [1,1]])
B = np.array([[-11,-10,16], [7,8,-11]])/9 #left inverse of A
C = np.array([[0,-1,6], [0,1,-4]])/2 #Another left inverse of A
#Let's check
B @ A
```

Out[ ]:
```
array([[ 1.0000000e+00,  0.0000000e+00],
       [-4.4408921e-16,  1.0000000e+00]])
```

In [ ]:
```
C @ A
```

Out[ ]:
```
array([[1., 0.],
       [0., 1.]])
```

11.2. Inverse

If `A` is invertible, its inverse is given by `np.linalg.inv(A)`. You'll get an error if `A` is not invertible, or not square.

In [ ]:
```
A = np.array([[1,-2,3], [0,2,2], [-4,-4, -4]])
B = np.linalg.inv(A)
B
```

Out[ ]:
```
array([[-2.77555756e-17, -5.00000000e-01, -2.50000000e-01],
       [-2.00000000e-01,  2.00000000e-01, -5.00000000e-02],
       [ 2.00000000e-01,  3.00000000e-01,  5.00000000e-02]])
```

In [ ]:
```
B @ A
```

Out[ ]:
```
array([[ 1.00000000e+00, -2.22044605e-16, -2.22044605e-16],
       [ 0.00000000e+00,  1.00000000e+00,  8.32667268e-17],
       [ 0.00000000e+00,  5.55111512e-17,  1.00000000e+00]])
```

In [ ]:
```
A @ B
```

Out[ ]:
```
array([[ 1.00000000e+00,  0.00000000e+00, -1.38777878e-17],
       [ 5.55111512e-17,  1.00000000e+00,  1.38777878e-17],
       [ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00]])
```

Dual basis. The next example illustrates the dual basis provided by the rows of the inverse of $B = A^{-1}$. We calculate the expansion

$$x = (b_1^T x)a_1 + \cdots + (b_n^T x)a_n$$

for a 3×3 example (see page 205 of VMLS).

In [ ]:
```
A = np.array([[1,0,1], [4,-3,-4], [1,-1,-2]])
B = np.linalg.inv(A)
x = np.array([0.2,-0.3,1.2])
RHS = (B[0,:]@x) * A[:,0] + (B[1,:]@x) * A[:,1] + (B[2,:]@x) *
↪  A[:,2]
print(RHS)
```

```
[ 0.2 -0.3  1.2]
```

Inverse via QR factorization. The inverse of a matrix A can be computed from its QR factorization $A = QR$ via the formula $A^{-1} = R^{-1}Q^T$.

In [ ]:
```
A = np.random.normal(size = (3,3))
np.linalg.inv(A)
```

Out[ ]:
```
array([[ 0.83904201, -1.17279605,  1.02812262],
       [ 1.28411762, -0.30441464,  0.9310179 ],
       [ 0.06402464, -0.0154395 ,  1.51759022]])
```

In [ ]:
```
Q,R = QR_factorization(A)
np.linalg.inv(R) @ Q.T
```

Out[ ]:
```
array([[ 0.83904201, -1.17279605,  1.02812262],
       [ 1.28411762, -0.30441464,  0.9310179 ],
```

```
       [ 0.06402464, -0.0154395 ,  1.51759022]])
```

11.3. Solving linear equations

Back substitution. Let's first implement back substitution (VMLS Algorithm 11.1) in Python, and check it.

In [ ]:
```
def back_subst(R,b_tilde):
    n = R.shape[0]
    x = np.zeros(n)
    for i in reversed(range(n)):
        x[i] = b_tilde[i]
        for j in range(i+1,n):
            x[i] = x[i] - R[i,j]*x[j]
        x[i] = x[i]/R[i,i]
    return x
R = np.triu(np.random.random((4,4)))
b = np.random.random(4)
x = back_subst(R,b)
np.linalg.norm(R @ x - b)
```

Out[ ]:
```
1.1102230246251565e-16
```

The function `np.triu()` gives the upper triangular part of a matrix, *i.e.*, it zeros out the entries below the diagonal.

Solving system of linear equations. Using the `gram_schmidt`, `QR_factorization` and `back_subst` functions that we have defined in the previous section, we can define our own function to solve a system of linear equations. This function implements the algorithm 12.1 in VMLS.

In [ ]:
```
def solve_via_backsub(A,b):
    Q,R = QR_factorization(A)
    b_tilde = Q.T @ b
    x = back_subst(R,b_tilde)
    return x
```

This requires you to include the other functions in your code as well.