# 10. Matrix multiplication

## 10.1. Matrix-matrix multiplication

In Python the product of matrices `A` and `B` is obtained with `A @ B`. Alternatively, we can compute the matrix product using the function `np.matmul()`. We calculate the matrix product on page 177 of VMLS.

In [ ]:
```
A = np.array([[-1.5, 3, 2], [1, -1,0]]) #2 by 3 matrix
B = np.array([[-1,-1], [0,-2], [1,0]]) #3 by 2 matrix
C = A @ B
print(C)
```

```
[[ 3.5 -4.5]
```

In [ ]:
```
 [-1.   1. ]]
 C = np.matmul(A,B)
print(C)
```

```
[[ 3.5 -4.5]
 [-1.   1. ]]
```

**Gram matrix.** The Gram matrix of a matrix $A$ is the matrix $G = A^T A$. It is a symmetric matrix and the $i, j$ element $G_{i,j}$ is the inner product of columns $i$ and $j$ of $A$.

In [ ]:
```
A = np.random.normal(size = (10,3))
G = A.T @ A
print(G)
```

```
[[ 8.73234113 -2.38507779  1.59667064]
 [-2.38507779  3.07242591  0.49804144]
 [ 1.59667064  0.49804144 15.65621955]]
```

In [ ]:
```
#Gii is norm of column i, squared
G[1,1]
```

Out[ ]:
```
3.072425913055046
```

In [ ]:
```
np.linalg.norm(A[:,1])**2
```

Out[ ]:
```
3.072425913055046
```

In [ ]:
```
#Gij is inner product of columns i and j
G[0,2]
```

Out[ ]:
```
1.5966706354935412
```

In [ ]:
```
A[:,0] @ A[:,2]
```

Out[ ]:
```
1.5966706354935412
```

**Complexity of matrix triple product.** Let's check the associative property, which states that $(AB)C = A(BC)$ for any $m \times n$ matrix $A$, any $n \times p$ matrix $B$, and any $p \times q$ matrix $C$. At the same time we will see that the left-hand and right-hand sides take very different amounts of time to compute.

In [ ]:
```
import time
m = 2000
n = 50
q = 2000
p = 2000
A = np.random.normal(size = (m,n))
B = np.random.normal(size = (n,p))
C = np.random.normal(size = (p,q))
start = time.time()
LHS = (A @ B) @ C
end = time.time()
print(end - start)
```

```
0.17589497566223145
```

In [ ]:
```
start = time.time()
LHS = (A @ B) @ C
end = time.time()
print(end - start)
```

```
0.23308205604553223
```

In [ ]:
```
start = time.time()
RHS = A @ (B @ C)
end = time.time()
print(end - start)
```

```
0.026722192764282227
```

In [ ]:
```
start = time.time()
RHS = A @ (B @ C)
end = time.time()
print(end - start)
```

```
0.025452852249145508
```

In [ ]:
```
np.linalg.norm(LHS - RHS)
```

Out[ ]:
```
4.704725926477414e-10
```

In [ ]:
```
start = time.time()
D = A @ B @ C      #Evaluated as (A@B)@C or as A@(B@C)?
end = time.time()
print(end - start)
```

```
0.24454998970031738
```

From the above, we see that evaluating `(A@B)@C` takes around 10 times as much time as evaluating `A@(B@C)`, which is predicted from the complexities. In the last line, we deduce that `$A@B@C$` is evaluated left to right, as `$(A@B)@C$`. Note that for these particular matrices, this is the (much) slower order to multiply the matrices.

## 10.2. Composition of linear functions

**Second difference matrix.** We compute the second difference matrix on page 184 of VMLS.

In [ ]:
```
D = lambda n: np.c_[-np.identity(n-1), np.zeros(n-1)] +
↪  np.c_[np.zeros(n-1), np.identity(n-1)]
D(5)
```

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