FLOATING POINT ARITHMETIC 1

In [ ]:
```
import numpy as np
```

In [ ]:
```
# Compute the Machine epsilon such that fl(1 + eps) > 1.
machine_epsilon = np.float32(1)
while np.float32(1) + machine_epsilon != np.float32(1):
    last_machine_epsilon = machine_epsilon
    machine_epsilon = np.float32(machine_epsilon) / np.float32(2)

print(last_machine_epsilon)
```

```
1.1920929e-07
```

In [ ]:
```
# We can define a function that computes the Machine epsilon for the
# wanted arithmetic
def machine_epsilon_computation(f=float):
    machine_epsilon = f(1)
    while f(1) + machine_epsilon != f(1):
        last_machine_epsilon = machine_epsilon
        machine_epsilon = f(machine_epsilon) / f(2)
    return last_machine_epsilon

print(f'Machine epsilon double: {machine_epsilon_computation(np.float32)}')
print(f'Machine epsilon double: {machine_epsilon_computation(np.float64)}')
```

```
Machine epsilon double: 1.1920928955078125e-07
Machine epsilon double: 2.220446049250313e-16
```

FLOATING POINT ARITHMETIC 2

In [ ]:
```
# We want to see the development of a_n during n iteration, it goes approximatel
# to euler number
ns = [10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000, 10
for n in ns:
    a_n = (1 + (1/n))**n
    print(np.e - a_n)


# The sequence reaches the best approximation at 10^8 iterations
```

```
0.12453936835904278
0.013467999037516609
0.0013578962234515046
0.00013590163411958434
1.359126674760347e-05
1.359363291708604e-06
1.3432696333026684e-07
3.011168736577474e-08
-2.2355251516614771e-07
-2.2477574246337895e-07
```

FLOATING POINT ARITHMETIC 3

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

```python
eig_A = np.linalg.eigvals(A)
rank_A = np.linalg.matrix_rank(A)
print(eig_A)
print(f'Rank(A) = {rank_A}')
eig_B = np.linalg.eigvals(B)
rank_B = np.linalg.matrix_rank(B)
print(eig_B)
print(f'Rank(B) = {rank_B}')
print("\n")

# We can see that A is a full-rank matrix and B is not
# The fact that the rank of a matrix is not full is correlated with the fact
# that the matrix has one or more eigenvalues that are 0.
# The rank of a matrix is computed as its dimension - the number of
# eigenvalues that are 0
```

```
[5. 2.]
Rank(A) = 2
[5. 0.]
Rank(B) = 1
```

In [ ]:

```python
# Other examples
C = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 0]])
eig_C = np.linalg.eigvals(C)
rank_C = np.linalg.matrix_rank(C)
print(eig_C)
print(f'Rank(C) = {rank_C}')
```

```
[1. 1. 0.]
Rank(C) = 2
```